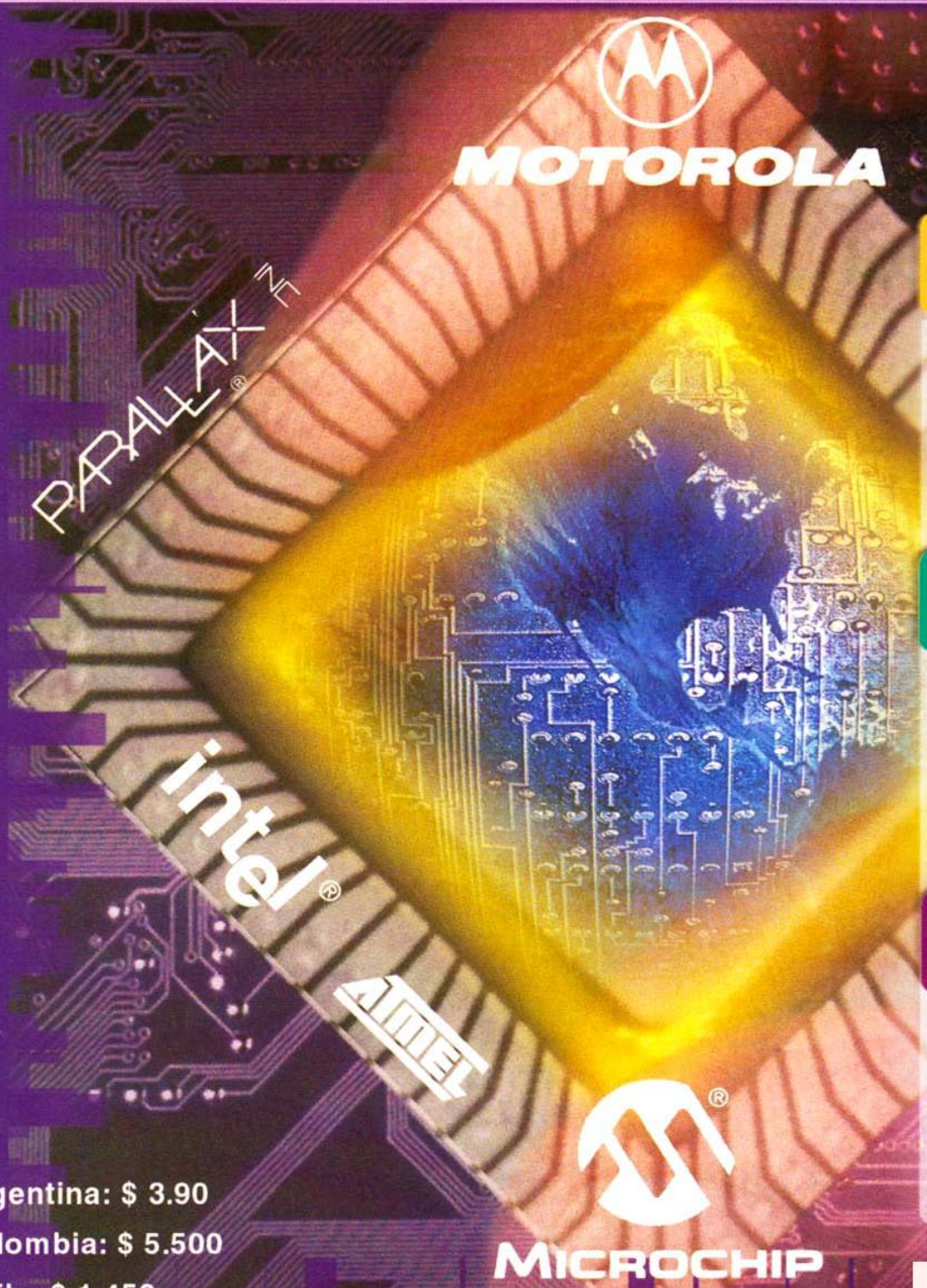
1

CEKIT S.A.

Curso práctico sobre

MICROCONTROLADORES

Teoría, Programación, Diseño, Prácticas y Proyectos completos



Teoría

Los microprocesadores
Evolución histórica

Programación

Conceptos básicos

Lenguajes de programación

Proyectos

Introducción

Pasos a seguir para elaborar un proyecto

Argentina: $ 3.90
Colombia: $ 5.500
Chile: $ 1.450
México: $ 27
Venezuela: Bs. 2.000

Tomo 1
- Teoría -

Curso práctico sobre

MICROCONTROLADORES

Teoría, Programación, Diseño, Prácticas y Proyectos completos

El **Curso práctico sobre MICROCONTROLADORES** de CEKIT S. A. se publica en forma de 40 fascículos de aparición semanal, encuadernables en 3 volúmenes. Cada fascículo consta de 4 páginas de cubiertas y 20 páginas de contenido a todo color. De estas últimas, 8 están dedicadas a la sección de **TEORIA**, 4 a **PROGRAMACIÓN** y 8 a la sección de **PROYECTOS.**

Las páginas de cada sección son encuadernables por separado. Por esta razón, al formar los volúmenes, debe cuidarse de desprender previamente, de todos los fascículos, las 8 páginas centrales de **PROYECTOS**, las 4 de **PROGRAMACIÓN** y por último las 8 páginas de **TEORÍA**. Durante la publicación de los fascículos se pondrán a la venta las tapas de los 3 volúmenes de la obra para su encuadernación.

CEKIT S.A. garantiza la publicación de la totalidad de la obra, el suministro de la tapas necesarias para su encuadernación y el servicio de números atrasados hasta un año después de terminada la circulación del último fascículo. También garantiza la fidelidad de la información teórico-práctica suministrada y el correcto funcionamiento de los programas incluidos como ejemplos de la obra.

Motorola, Microchip, PIC, Intel, Atmel, Parallax, Zilog, Phillips, son marcas registradas de las diferentes empresas en los Estados Unidos y en otros paises. Otros productos y nombres de compañías mencionadas en esta obra pueden ser marcas registradas de sus propietarios.

Este curso ha sido elaborado según el plan del editor y de sus autores, y bajo su responsabilidad, por los siguientes integrantes del departamento técnico de CEKIT S.A.

Distribución Internacional

Argentina: Importador: Editorial Conosur S.A. Av. Belgrano 355 10º. Tel: 4342-9029 Fax: 4342-9025 e-mail: gconosur@speedy.com.ar Dist. Capital: VACCARO SANCHES Y CIA. - Moreno 794, 9º(1092) Buenos Aires. Dist. Interior: DISTRIBUIDORA BERTRAN S.A.C. - Av. Vélez Sarsfield 1950(1285) Buenos Aires. - Bolivia: AGENCIA MODERNA LTDA. - Chile: DISTRIBUIDORA ALFA S.A. - Colombia: DISTRIBUIDORAS UNIDAS. - Venezuela: DISTRIBUIDORA CONTINENTAL.- México: CONSORCIO SAYROLS - Ecuador: DISTRIBUIDORA ANDES. - Panamá: PANAMEX S.A. - Perú: DISTRIBUIDORA BOLIVARIANA S.A. - Paraguay: SELECCIONES S.A.C. - Uruguay: DISTRIBUIDORA CAREAGA

Suscripciones (no válidas en Argentina), números atrasados y consultas técnicas en Colombia

CEKIT S.A.
Calle 22 Nº 8 - 22, Piso 2, A.A. 194 Pereira(Colombia)
o a nuestro correo electrónico:
ecekit@col2.telecom.com.co

ISBN (Fascículo 8):958-657-510-1
ISBN (Volumen 1):958-657-498-9
ISBN (Obra completa): 958-657-497-0

CEKIT S.A.®
Capacitamos para vivir mejor

Calle 22 Nº 8 - 22, Piso 2,
Teléfonos
(6) 3348179- 3348187-
3348189 - 3348193
Fax: (6) 3348020
Pereira-Colombia
Carrera 13 Nº 38 - 43, Piso 1.
Teléfonos: 2873086 -
2873420 - 2877318
Bogotá - Colombia

e-mail: ecekit@col2.telecom.com.co
http://www.cekit.com.co

Gerente General:
Jorge Hernán Alvarez
Gerente Comercial:
Marcelo Alvarez H.
Director Editorial:
Manuel Felipe González G.
Circulación Internacional:
Humberto Real Blanco.

Autores

JUAN ANDRÉS CASTAÑO W.
MANUEL FELIPE GONZÁLEZ G.
GILBERTO VARGAS C.
JUAN DIEGO HERNÁNDEZ F.

Proyectos

JOHN JAIME ROBBY G.

Diseño carátula: Luz Angela Plata B.
Diseño: Gloria Marcela Gómez G.
Diagramación: Héctor Hugo Jiménez

© CEKIT S.A. 2001
Pereira - Colombia

Todos los derechos reservados. Prohibida su reproducción parcial o total por cualquier medio sin permiso escrito del editor.

EDICIÓN CONO SUR
Internacional Graphics & Printing Co.
2600 Douglas Road, Suite 406
Coral Gables, Florida 33134, U.S.A.

Impreso y encuadernado por:
RR Donnelley
América Latina
Ruta Panamericana Km. 38.700
Buenos Aires Argentina
Impreso en Argentina
Junio 2002

Curso práctico sobre

MICROCONTROLADORES

Teoría, Programación, Diseño, Prácticas y Proyectos completos

CEKIT S.A.

Estudie, con el método de CEKIT de "Aprender haciendo", la tecnología de los Microcontroladores, una de las que más han cambiado el mundo en los últimos años y gane mucho dinero. Estos pequeños cerebros electrónicos se encuentran en todas partes: automóviles, computadoras, electrodomésticos, alarmas, juegos de luces, instrumentos, equipos de comunicaciones, etc.

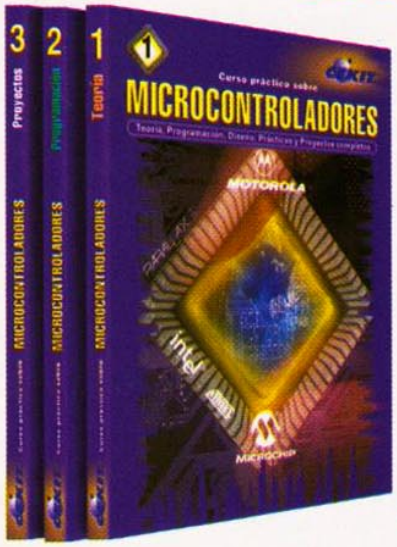
En este curso incluimos los principios teóricos, los fundamentos de su programación, una gran cantidad de actividades prácticas y proyectos completos para ensamblar.

Los microcontroladores son los elementos básicos y fundamentales de una nueva revolución, la interconexión de todo con todo en una red mundial de información y control. Esto permitirá la comunicación de hechos y datos entre personas y objetos formando la red del conocimiento, base de la economía del nuevo milenio **Capacítese y forme parte activa de la nueva economía.**

Libere su creatividad

Las personas con buenos conocimientos de esta tecnología, tendrán muchas oportunidades laborales o podrán fabricar sus propios aparatos en forma industrial, lo que puede ser la base para una empresa muy rentable.

Como está estructurado el curso



El **Curso práctico sobre Microcontoladores** de CEKIT S.A. se publicará en 40 fascículos con tres secciones cada uno, y al final, se agrupará en tres tomos de pasta dura con cada uno de los temas principales. Para su fácil identificación, cada una de las secciones tiene un color diferente. La distribución de los tomos es la siguiente:

Tomo 1: Teoría	Tomo 2: Programación	Tomo 3: Proyectos
320 páginas (1 a 320)	160 páginas (1 a 160)	320 páginas (1 a 320)

Sección de Teoría

Los siguientes son los principales temas tratados en esta sección:

- Introducción a los microprocesadores
- Introduccion a los microcontroladores
- Diferencias entre un microprocesador y un microcontrolador
- Principales familias de microcontroladores
- Estuctura básica de los sistemas con microcontrolador
- (Motorola y PIC de Microchip)
- Arquitecturas de microcontroladores
- La CPU y la ALU
- Los registros
- Organización de la memoria
- La EPROM de datos
- Las interrupciones
- Los puertos de entrada/salida (I/O)
- Los temporizadores (*Timers*)
- Las opciones Comparación/Captura/PWM
- El puerto serial síncrono SSP, bssp
- La USART
- Las referencias de voltaje
- Los comparadores
- Los conversores A/D
- Modos de bajo consumo (*sleep* y *watchdog*)
- Bits de configuracion
- Programación en el circuito (*in circuit*)
- El conjunto (*set*) de instrucciones
- Las comunicaciones seriales
- Otros microcontroladores
- Parallax (Basic Stamp)
- Intel
- Atmel

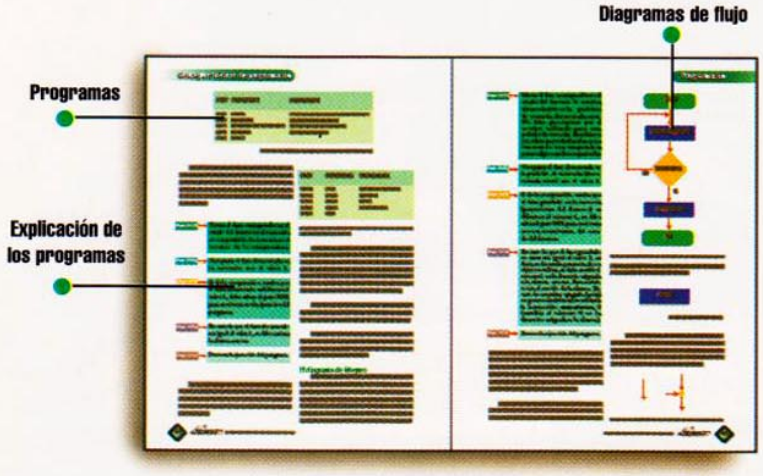
En la sección de teoría, se estudian los fundamentos teóricos y funcionales de los microcontroladores. Esto se refiere a la estructura o arquitectura interna de los mismos, así como a la función de cada uno de los terminales o pines de los diferentes dispositivos tratados en el curso cuyo conocimiento es necesario para diseñar circuitos con estos componentes.

Se empieza con un repaso sobre el tema de los microprocesadores, los cuales dieron origen a los microcontroladores, y se desarrollan durante las lecciones, los diferentes aspectos técnicos de esta tecnología.

Sección de Programación

Todo aparato o proyecto con microcontroladores incluye, además de los diferentes circuitos, uno o varios programas que se deben encribir en el lenguaje apropiado según el tipo de microcontrolador utilizado. Este curso le proporcionará los conocimientos necesarios para que adquiera la habilidad de programarlos.

Los principales temas tratados son:

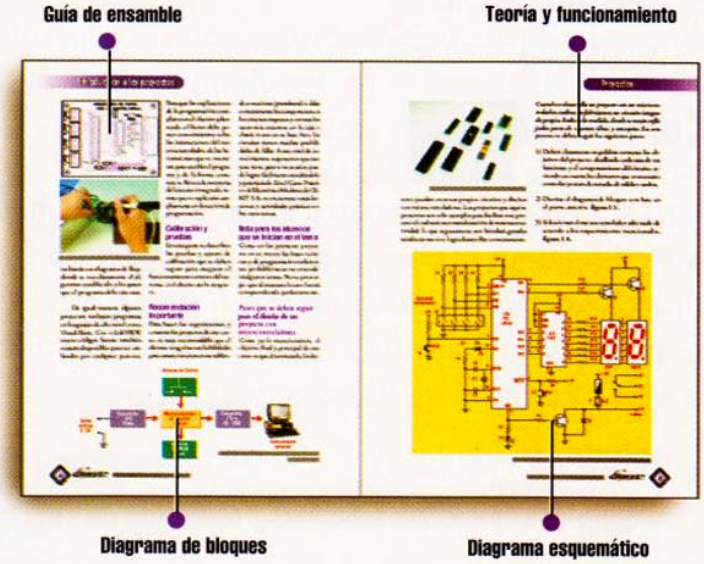


- Programación de un microcontrolador
- El ciclo de desarrollo de un programa
- Herramientas de *software*
- Programación en modo inmediato
- Programación orientada a registros
- Control de flujo de un programa
- Puertos de entrada/salida
- Subrutinas
- Interrupciones
- Programación en lenguaje estructurado
- Temporizadores
- Programación serial
- Programando los conversores A/D y D/A
- Programación en lenguaje C
- Tópicos avanzados de programación

Sección de proyectos

- Indicador binario de 4 bits
- Alarma para el hogar y el automóvil
- Reloj digital y calendario
- Secuenciador de luces programable
- Cerradura codificada
- Activación de aparatos vía telefónica
- Temporizador universal
- Aviso luminoso con diodos LED
- Luces rítmicas
- Convertidor paralelo-serial
- Punta lógica
- Temporizador miniatura
- Frecuencímetro
- Generador de señales
- Programador de memorias
- Adquisicion de datos en 16 bits
- Minirobot con microcontrolador
- Adquisición de datos por el puerto serial
- Tarjeta I/O para el puerto serial
- y muchos otros...

En esta importante sección del curso, se entrega una amplia información con la cual el alumno podrá ensamblar una buena cantidad de proyectos. Para cada uno de ellos se incluye la teoría de funcionamiento, el diagrama esquemático, la lista de materiales, las instrucciones para el ensamblaje y el listado del programa.



A quienes va dirigido el curso

El tema de los microcontroladores es de gran utilidad para un grupo muy amplio de personas que requieran conocer esta tecnología por su gran cantidad de aplicaciones.

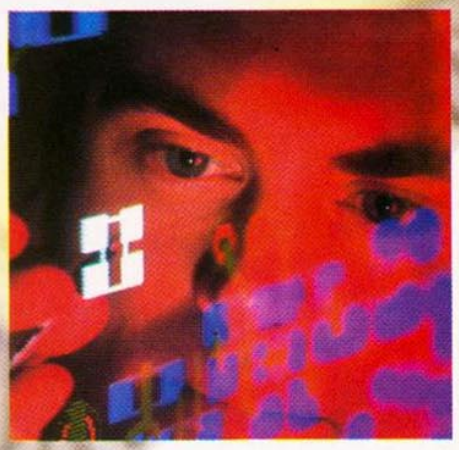
Los principales grupos de usuarios son:

Estudiantes y profesionales de Ingeniería y Tecnologías:

- Electrónica
- Eléctrica
- Mecánica
- Sistemas
- Bioingeniería
- Mecatrónica
- Automatización
- Estudiantes de colegios técnicos con énfasis en electrónica

Técnicos en:

- Electrónica industrial
- Automatización
- Electrónicos en general
- Alumnos de los cursos de CEKIT S.A.
- Estudiantes y profesionales en FISICA, para sus necesidades de instrumentación



Autodidactas o aficionados con conocimientos en:

- Electrónica Digital
- Microprocesadores

Certificado

Una vez terminado el curso y aprobada la evaluación que se publicará al finalizar la obra, el alumno recibirá un Certificado de Aptitud Ocupacional otorgado por CEKIT S.A. y aprobado por las respectivas autoridades educativas. Este certificado ler permitirá acceder a puestos calificados de trabajo.

Otra obra con la calidad editorial y didáctica de CEKIT S.A.®

Capacitamos para vivir mejor

Es muy placentero y satisfactorio para CEKIT S.A. presentar a todos nuestros lectores y amigos, estudiantes, profesores, aficionados, técnicos, profesionales y en general a la comunidad técnica internacional, una nueva obra didáctica en el campo de la electrónica con el estilo propio y la metodología de "Aprender haciendo" que nos ha caracterizado por tantos años. Se trata del *Curso práctico sobre MICROCONTROLADORES*, el cual tiene como objetivo principal dar a conocer y enseñar, de una manera fácil y agradable, esta maravillosa tecnología.

Los microcontroladores, que pueden considerarse como una microcomputadora en un sólo empaque, han permitido un avance sin par en los últimos años en el diseño y la fabricación de todo tipo de aparatos y aplicaciones electrónicas. Haciendo un símil con el ser humano, podemos decir que un microcontrolador es un pequeño cerebro electrónico formado por muchos transistores interconectados entre sí tal como las neuronas y que tiene varias zonas con funciones especializadas como la memoria, uno o varios circuitos para recibir información de teclados, interruptores y sensores, entre otros, y secciones para activar elementos externos como motores, relés, indicadores sonoros, pantallas, etc.; todo esto coordinado por medio de un programa o conjunto de instrucciones.

Nunca antes, en la historia de la electrónica, un componente había permitido tanta creatividad y desarrollo de productos de una forma tan fácil, en tan corto tiempo y a costos tan reducidos. Con los microcontroladores es posible diseñar y fabricar casi cualquier aparato que se nos ocurra; el límite está dado sólo por la imaginación y nuestra capacidad de hacerlo.

Aunque el tema parece difícil y complejo, realmente no lo es y para demostrar lo contrario, lo iremos explicando paso a paso en un lenguaje simple acompañado de muchas figuras, diagramas e ilustraciones de tal forma que los lectores lo puedan ir asimilando a su propio ritmo de aprendizaje sin mayores dificultades. Para entender y aplicar esta tecnología, se requieren conocimientos de Electrónica Digital, de los sistemas de numeración, especialmente del sistema binario y hexadecimal, y nociones básicas sobre los microprocesadores. Sin embargo, durante el curso repasaremos algunos de los conceptos fundamentales de estos temas con el fin de que los lectores que no los conozcan, puedan comprender mejor cada una de las lecciones.

La mayoría de los aparatos electrónicos modernos incluyen uno o varios microcontroladores y de ahí la importancia y necesidad de este curso en este mundo tecnológico en el cual nos encontramos. Un reloj digital de pulsera lo utiliza para contabilizar el tiempo, mostrarlo en una pequeña pantalla y recibir las órdenes en los botones para ejecutar diferentes funciones. Una videograbadora lo utiliza para coordinar sus principales funciones como el encendido y el apagado, el cambio de los canales, el avance y retroceso de la cinta, las funciones del control remoto, etc. En un horno microondas el microcontrolador se encarga de recibir las órdenes que le damos en el teclado, mostrarlas en la pantalla y accionar el elemento que produce el calor en los alimentos.

Pero sin restarle importancia a este tipo de aplicaciones, es en otro campo a donde queremos orientar este curso. Se trata de la utilización de los microcontroladores en sistemas de control y en el diseño de aparatos aparentemente sencillos pero con una gran cantidad de usos en la vida diaria ya sea en el hogar, en la industria, en la oficina, en el campo o a nivel experimental y sobre todo para enseñarles a nuestros lectores cómo desarrollar su propia aplicación partiendo de sus necesidades particulares.

Para enseñar y explicar mejor esta tecnología, hemos dividido este curso en varias secciones así:

- Una sección de **Teoría** en donde se presentan los conceptos básicos de los microcontroladores iniciando con su origen basado en los microprocesadores, su estructura o arquitectura interna, su funcionamiento eléctrico y lógico, la operación de sus bloques internos, etc. En fin, toda la información necesaria para poder configurar un circuito con un microcontrolador y entender su funcionamiento.

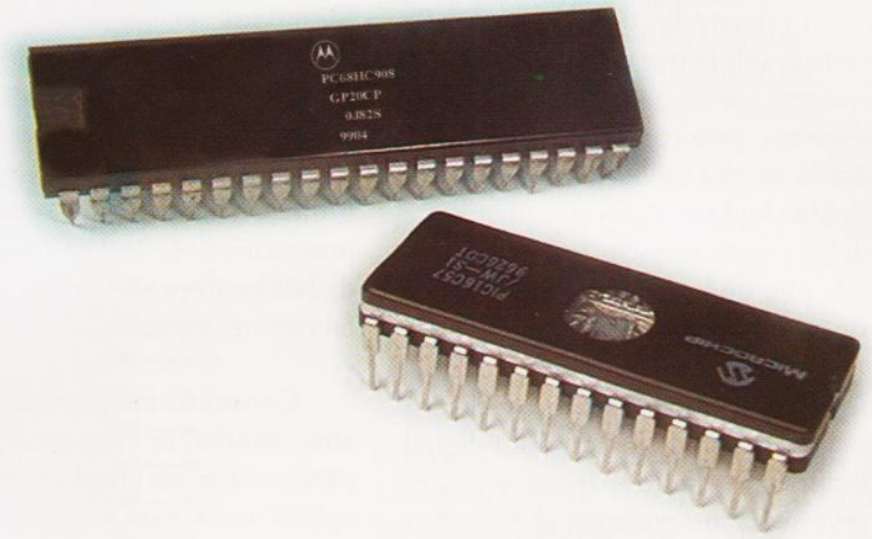
- Una sección de **Programación** en donde se explicará qué es un programa de microcontrolador, los diferentes lenguajes que se utilizan, las técnicas adecuadas para elaborar programas eficientes, cómo se lleva al interior del microcontrolador y las principales herramientas de software y hardware para elaborarlos. Esta es una sección de mucha importancia dentro del curso ya que la programación es donde radica la mayor dificultad de muchas personas para acceder a esta tecnología y por ello, dedicaremos un gran esfuerzo didáctico para que todos nuestros alumnos puedan escribir sus propios programas de una forma correcta.

- Y para completar la estructura del curso, de acuerdo a la metodología de CEKIT S.A., hay una seccción de **Proyectos** y actividades prácticas en donde se presentan ejemplos completos con aplicaciones de diferentes tipos de microcontroladores. En esta sección también se incluirán los diagramas de pines y las características de los microcontroladores más representativos de las principales marcas o fabricantes y una explicación de los componentes y circuitos que los acompañan, desde un simple LED, hasta pantallas de cristal líquido (LCD), sensores, conversores A/D, acondicionadores de señal, etc.

Aunque existen en el mercado una buena cantidad de fabricantes de microcontroladores, cada uno con una gran variedad de modelos, hemos seleccionado como ejemplos para este curso principalmente los de Motorola y Microchip (PIC) por ser los de mayor difusión y aplicación actualmente; con un menor cubrimiento, los de INTEL, ATMEL, PARALLAX y otros, los cuales esperamos tratar en un próximo curso.

Felipe González G.
Editor General

LECCIÓN 1



INTRODUCCIÓN A LOS MICROCONTROLADORES

Un microcontrolador (*microcontroller*, en inglés) es un circuito integrado que contiene toda la estructura de una microcomputadora, o sea, CPU (Unidad Central de Proceso), memoria RAM , memoria ROM, circuitos de entrada-salida (I/O) y otros módulos con aplicaciones especiales. Su nombre nos indica sus principales características: micro por lo pequeño y controlador porque se utiliza principalmente para controlar otros circuitos o dispositivos eléctricos, mecánicos, etc.

Este maravilloso componente resume, en un solo circuito integrado, más de cuarenta años de tecnología electrónica. Es difícil encontrar en la actualidad un aparato o una aplicación que no lo utilice o no pueda utilizarlo para optimizar su operación.

Se ha mencionado y confirmado muchas veces que la electrónica es la tecnología de los componentes. El desarrollo de cada nuevo dispositivo de estado sólido trae consigo técnicas de diseño diferentes, por lo general, más simples. En los años sesenta, para construir un reloj digital, era necesario acoplar un buen número de circuitos integrados lógicos. Al mismo tiempo, el diseñador debía poseer conocimientos muy claros sobre cada uno de los elementos, para realizar, con éxito, la integración.

A partir de 1970, el panorama de la electrónica y especialmente el de la electrónica digital, cambió radicalmente cuando apareció en el mercado un nuevo supercomponente: el microprocesador. Esto introdujo un concepto novedoso que en la actualidad se conserva y refuerza cada vez más, el de la ***lógica programada***. Antes de los microprocesadores, los circuitos electrónicos se diseñaban para una función específica la cual no podía modificarse sin cambiar físicamente las conexiones, el número y la cantidad de los diferentes elementos que los formaban a lo cual se le llamó ***la lógica cableada***, **figura 1.1**.

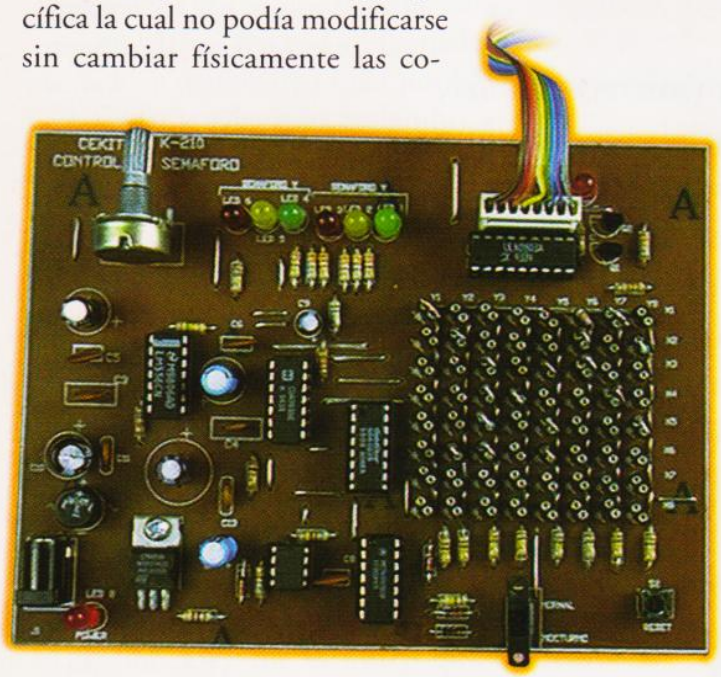


Figura 1.1. *Ejemplo de un circuito con lógica cableada*

Con los microprocesadores y ahora con los circuitos lógicos programables (FPGA), los microcontroladores y los circuitos de DSP (Procesamiento Digital de señales), entre otros, es posible modificar cuantas veces se requiera el programa, aún en forma remota por la internet, y el circuito o aparato en el cual están instalados, trabajará de una forma completamente diferente sin modificar físicamente una sola conexión. De esta forma, el límite de los diseños solo lo establece la imaginación de los programadores y la capacidad de los dispositivos que lancen al mercado los fabricantes especializados en estas tecnologías.

Consolidadas las técnicas digitales de los años sesenta, se creó entonces la necesidad de profundizar en el estudio y desarrollo de las aplicaciones para los microprocesadores y la programación en lenguaje de máquina o *assembler*. Fué la época de oro del 8080, el 8086, el Z-80, el 6809, el 6502, el 68000 y otros microprocesadores, utilizados como circuitos centrales en las aplicaciones de control.

En 1980, aproximadamente, los fabricantes de circuitos integrados iniciaron la difusión de un nuevo circuito con aplicaciones para control, medición e instrumentación, al que llamaron "microcomputador en un sólo chip" o, de manera más exacta y concisa: ***microcontrolador***. Para entender mejor la estructura interna, el fucionamiento y los orígenes de los microcontroladores, repasaremos primero el tema de los microprocesadores.

Los microprocesadores

Desde el punto de vista funcional, un microprocesador es un circuito integrado, **figura 1.2**, que incorpora en su interior una unidad central de proceso (CPU) y todo un conjunto de elementos lógicos que per-

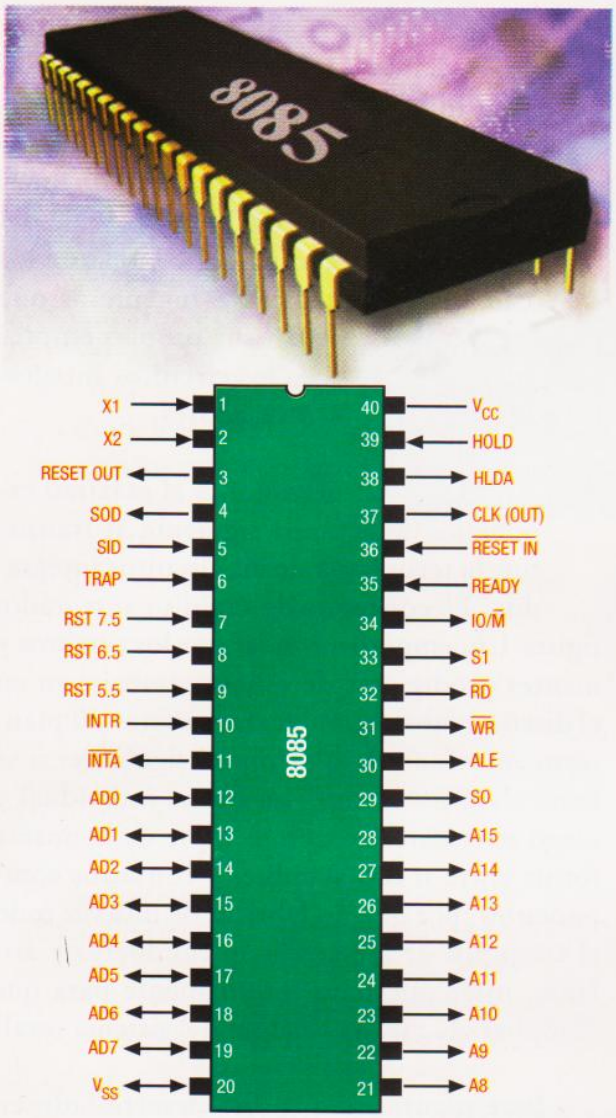


Figura 1.2. *Microprocesdor 8085*

miten enlazar otros dispositivos como memorias y puertos de entrada o de salida (I/O), formando un sistema completo para cumplir con una aplicación específica dentro del mundo real. Para que el sistema pueda realizar su labor debe ejecutar paso a paso un programa que consiste en una secuencia de números binarios o instrucciones, almacenadas en uno o más elementos de memoria, generalmente externos al mismo. La aplicación más importante de los microprocesadores y que cambió totalmente la forma de trabajar, ha sido la computadora personal o microcomputadora.

Un solo microprocesador, utilizado en el diseño de un sistema digital, puede reemplazar una gran cantidad de circuitos integrados de pequeña y mediana escala como compuertas, *flip-flops*, registros, contadores, decodificadores, memorias, sumadores, comparadores, etc, **figura 1.3**. Esto es ventajoso desde varios puntos de vista: facilidad de diseño, modularidad, versatilidad, flexibilidad, mantenimiento, bajo costo, confiabilidad, poco consumo de energía, etc.

Evolución histórica de los microprocesadores

Dentro de los anales de la electrónica está el año de 1970 como la fecha de invención del microprocesador. Desde entonces, este diminuto dispositivo, uno de los adelantos tecnológicos más importantes de nuestro siglo, ha revolucionado todos los campos de la vida humana. Con el microprocesador se inició una nueva era de desarrollo de la industria de las computadoras y de la electrónica, la cual hasta el presente ha ido evolucionando con una velocidad que aún sorprende incluso a los visionarios más futuristas. Nadie en esa época se imaginaba el impacto tan grande que causaría este desarrollo en la vida del hombre moderno.

Los aparatos electrónicos que utilizan microprocesadores, entre ellos las computadoras personales y de escritorio, **figura 1.4**, han cambiado totalmente la forma de trabajar e inves-

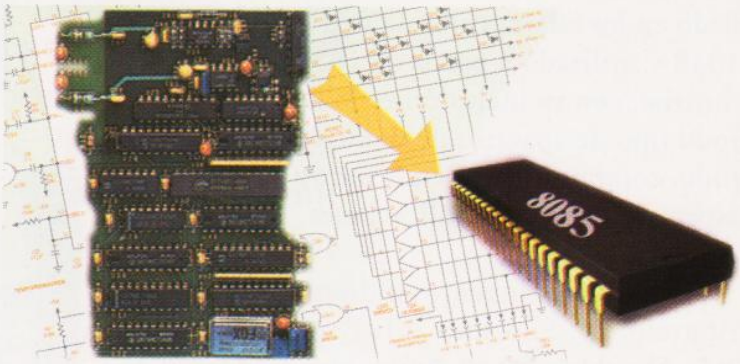


Figura 1.3. *Tarjeta con circuitos integrados individuales vs un Microprocesador*

Figura 1.4. *Computadora personal y su tarjeta principal*

tigar de los seres humanos. Ninguna herramienta anterior desarrollada por el hombre le había dado tanto poder para crear otras y acelerar su evolución en todos los campos. Para asimilar mejor el desarrollo de esta tecnología, retrocedamos un poco en la historia de la electrónica, sus componentes y aplicaciones.

Sin lugar a dudas, la senda la abrió el invento del tubo de vacío por Lee De Forest a comienzos del siglo, basado en el descubrimiento de un fenómeno llamado Efecto Edison. Este dispositivo hizo posible la radio, la telefonía, la telegrafía inalámbrica, etc., e impulsó el desarrollo comercial e industrial de la electrónica. Inclusive las primeras computadoras eran fabricadas con tubos de vacío. Luego vino la revolución del transistor, desarrollado en los laboratorios de Bell Telephone en 1948 y utilizado a partir de 1950 a escala industrial, con su inclusión en la fabricación de todo tipo de aparatos de radio, televisión, sonido, computadoras y en la industria militar y espacial, entre otras aplicaciones.

De los tres inventores del transistor (W. Brattain, W. Shockley y J. Bardeen), **figura 1.5**, Shockley fue el único que se dedicó a desarrollar métodos de fabricación en serie de este revolucionario dispositivo en su empresa, Shockley Semiconductor Laboratories, dando nacimiento a una nueva era en la industria electrónica y a la ubicación de esta tecnología en un sitio de California cercano a San Francisco que más tarde se llamó Silicon Valley o Valle del Silicio. En esta compañía se iniciaron los principales científicos e ingenieros que más tarde formarían sus propias empresas y desarrollarían los circuitos integrados y los microprocesadores.

El microprocesador es el máximo exponente de la etapa siguiente al transistor: la tecnología de los circuitos integrados. El concepto de circuito integrado, **figura 1.6**, empezó a rondar por los cerebros y mentes prodigiosas de quienes trabajaban en el diseño y fabricación de transistores. El planteamiento fue más o menos el siguiente: si se fabricaban transistores en forma individual y luego se tenían que unir siempre de la misma forma entre sí con alambres y con otros componentes ¿por qué no fabricar de una vez todo el conjunto de material semiconductor y aislante, interconectado internamente para que cumpliera la misma función del sistema total?

Este planteamiento fue desarrollado en la práctica simultáneamente, pero en forma independiente, por dos empresas muy im-

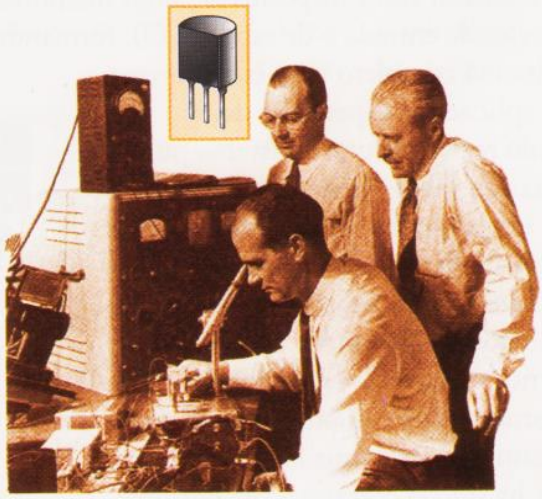


Figura 1.5. *El transistor y sus inventores*

portantes en la historia de la electrónica: Fairchild Semiconductor y Texas Instruments. En Fairchild dirigía el equipo de trabajo Robert Noyce, y en Texas Jack Kilby. Como anécdota, podemos mencionar que los primeros circuitos integrados salieron al mercado en el año de 1959 a un precio de 450 dólares y solamente incluían cuatro transistores y varias resistencias. Actualmente, un circuito integrado cuesta entre 10 y 20 centavos de dólar y un microcontrolador sencillo, con muchas más prestaciones, puede costar hasta un dólar o menos en escalas industriales. La explosión de los circuitos integrados desde el año 1960 hasta la fecha, ha permitido el desarrollo de la electrónica en una forma sorprendente.

Noyce renunció a la Fairchild en 1968 y fundó, en compañía de Gordon Moore y Andrew Grove y con el respaldo económico de Arthur Rock, la empresa Intel en donde se dieron los primeros pasos para el desarrollo del microprocesador. Esta compañía empezó a vislumbrar un gran mercado en el área de las computadoras y su investigación se orientó hacia el reemplazo de los circuitos de memoria magnéticos con núcleo de ferrita, por circuitos de memoria basados en semiconductores. La base técnica consistió en el uso de un simple *flip-flop* como elemento de memoria. Así se creó el primer circuito de memoria tipo RAM llamado el 1103, con una capacidad de 1024 bits.

El diseño del microprocesador se inició en un grupo de trabajo de Intel dirigido por Ted Hoff, un brillante ingeniero egresado de la Universidad de Stanford. Todo empezó cuando Intel se decidió a firmar un contrato con una compañía japonesa fabricante de calculadoras (la desaparecida Busicom Corporation). Ésta quería que se les fabricara un conjunto de circuitos integrados que reemplazaran la gran cantidad de componentes que tenían las calculadoras de ese entonces.

Después de un largo trabajo, se llegó hasta lograr que todo el circuito fuera reemplazado por tres chips, pero estos resultaron ser de un tamaño mayor de acuerdo a los requerimientos. A Hoff se le ocurrió que debía agrupar toda la parte de proceso aritmético y lógico en un solo circuito y el resto de la calculadora en los otros dos circuitos. Con la intervención de otro diseñador, Federico Faggin, el proyecto se llevó acabo con todo éxito. A este circuito, de 2250 elementos integrados en un área de 3 x 4 milímetros, se le llamó ***microprocesador***. También se le dio el nombre de CPU (*Central Processing Unit*) o MPU (*Micro Processing Unit*).

Aunque este circuito tenía ya muchas de las características de una unidad central de proceso integrada, el primer microprocesador en un sólo chip, fabricado como tal, fue el 4004 de Intel, diseñado para reemplazar grandes cantidades de circuitos integrados TTL. El 4004 era un chip muy sencillo que manipulaba datos de cuatro bits. Intel desarrolló muy pronto, en 1972, el 8008, el cual podía procesar datos de ocho bits. pero era muy lento.

Para remediar esto, Intel desarrolló un sustituto, el 8080, y posteriormente el 8085, com-

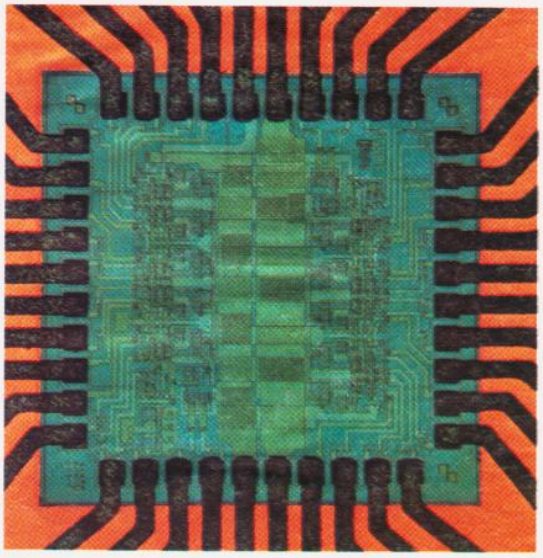


Figura 1.6. *Estructura interna básica de un circuito integrado*

patible con el primero, con funciones adicionales y menos circuitos de soporte. Un equipo de diseñadores que antes había trabajado para Intel en el 8080 formó la Zilog Inc. y construyó el microprocesador Z-80, el cual incorporaba un conjunto de instrucciones más extenso que el 8080, aunque era compatible con este último. Este microprocesador ha sido uno de los más utilizados en el campo de control.

Por la misma época en que se presentaba al mercado el 8080, otra empresa de semiconductores, Motorola, desarrolló el 6800, un microprocesador de 8 bits con un diseño completamente distinto pero con iguales características. Motorola perfeccionó el 6800 hacia el 6809, considerado como uno de los mejores procesadores de 8 bits de todas las épocas.

A pesar de sus excelentes características, el 6809 no tuvo el éxito comercial que se esperaba. Dicho éxito lo obtuvo otro derivado del 6800: el 6502, producido por MOS Technology. Con este microprocesador se fabricaron las primeras computadoras personales como la PET de Commodore y la Apple II de Apple Computer Inc.

A partir de ese momento, se estableció una guerra técnica y comercial, que aún subsiste, entre Intel y Motorola, la cual los ha llevado a ser los dos grandes líderes indiscutibles del mercado de los microprocesadores. Pero el desarrollo del microprocesador no se quedó ahí. A principios de la década de los 80, empezaron a aparecer los microprocesadores de dieciseis bits, mucho más potentes. El primero en salir al mercado fue el 8086 de Intel en 1978, el cual fue adoptado por la IBM para la fabricación de su famosa IBM PC. Lo siguieron de cerca el 68000, el 68020, el 68030 y el 68040 de Motorola. Con estos microprocesadores se inició en Apple una nueva familia de microcomputadoras: las Macintosh. Luego, en un consorcio entre Apple, IBM y Motorola se desarrolló una nueva familia de microprocesadores: los Power PC los cuales se utilizan en las computadoras Apple e IBM actuales.

Por los lados de Intel se desarrollaron el 8088 utilizado en el IBM XT, el 80186, el 80286 utilizado para la IBM AT, el 80386, el 80486 y los Pentium. Todos estos microprocesadores de Intel han sido utilizados para la fabricación de microcomputadoras de bajo costo llamadas clones, lo que ha hecho que esta empresa siga siendo el mayor fabricante de microprocesadores en el mundo.

La siguiente generación fueron los Pentium II, Pentium III y actualmente los Pentium IV con los cuales se han logrado velocidades de procesamiento impresionantes y una enorme capacidad de memoria, lo que los hace apropiados para moverse sin problemas en entornos como la internet, el video, la multimedia, el procesamiento de datos y el trabajo en tiempo real.

Los microprocesadores actuales

Dentro de los microprocesadores de la plataforma PC, actualmente podemos mencionar los productos de las firmas Intel, AMD, Cyrix y Digital. Las referencias más novedosas son el Pentium IV, el Althon, el 686 MMX y el Alpha, respectivamente, **figura 1.7**.

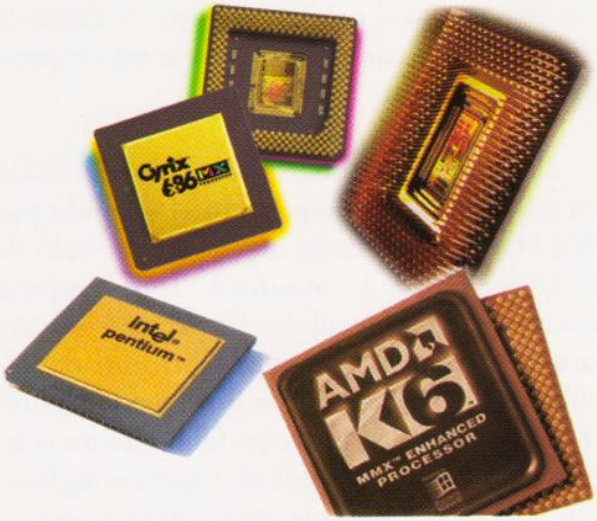
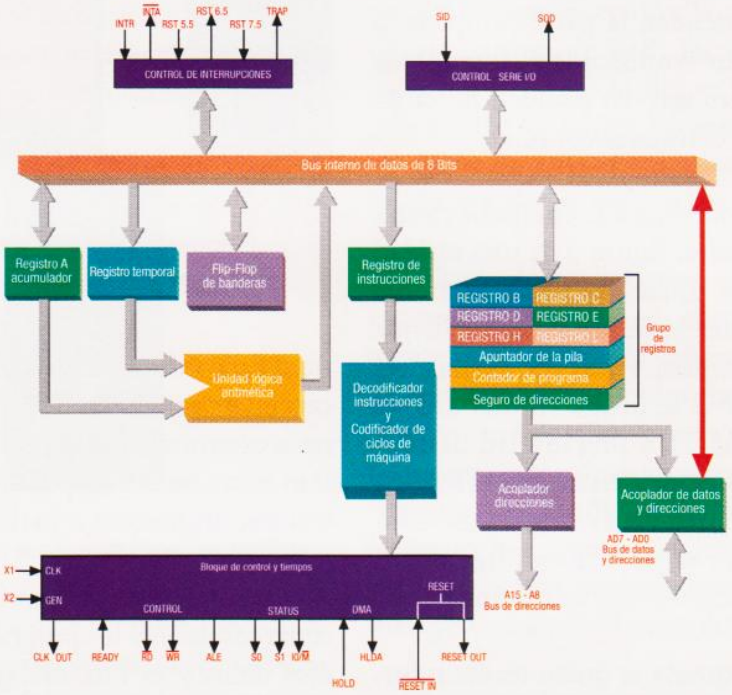


Figura 1.7. *Microprocesadores modernos*

LECCIÓN 2



LOS MICROPROCESADORES

Son los circuitos integrados que revolucionaron la electrónica al permitir el diseño de todo tipo de aparatos programables. Aunque su principal aplicación son las computadoras personales, también se encuentran en la mayoría de los circuitos de control.

El microprocesador, corazón de las computadoras

Una computadora digital es un equipo especializado en el procesamiento de información, cuyo principio de funcionamiento es relativamente sencillo, si lo vemos desde el punto de vista de los bloques que lo conforman. Cada uno de esos bloques está formado por varios circuitos integrados fáciles de entender, la parte compleja la encontramos cuando estudiamos el interior de dichos circuitos, pero esto en realidad no es de mucho interés para el usuario común.

Una computadora digital se compone de tres bloques fundamentales, **figura 2.1**: una unidad de entradas y salidas, una CPU y una memoria. A su vez, la unidad de entradas y salidas se divide en un bloque de entrada y otro de salida. Del mismo modo, la CPU se divide en una unidad aritmético-lógica (ALU), y una unidad de control y el bloque de memoria en memoria RAM y memoria ROM, obteniéndose seis bloques o unidades con funciones muy específicas, **figura 2.2**, cuyas funciones describiremos a continuación.

La unidad de entrada es quien recibe información del mundo exterior, ya sea de un operador humano o un fenómeno físico; esta unidad permite llevar esa información hacia la unidad de memoria para poder procesarla posteriormente; dispositivos de entrada son, por ejemplo, los teclados y el ratón. La unidad de salida entrega los resultados del procesamiento al mundo exterior. Esta información se lleva desde la unidad central de proceso o CPU, hacia circuitos o aparatos externos llamados periféricos que se encuentran fuera de la computadora, por ejemplo monitores, impresoras o parlantes.

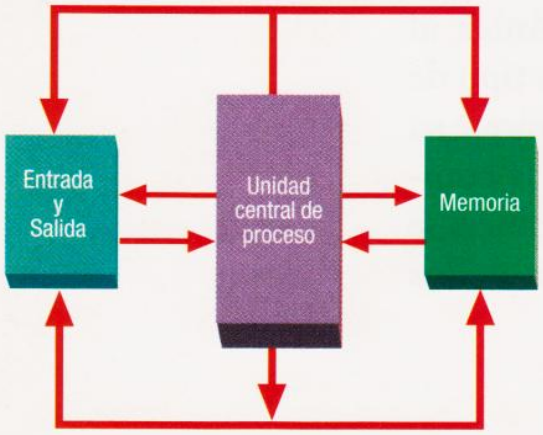


Figura 2.1 *Bloques principales de una computadora digital*

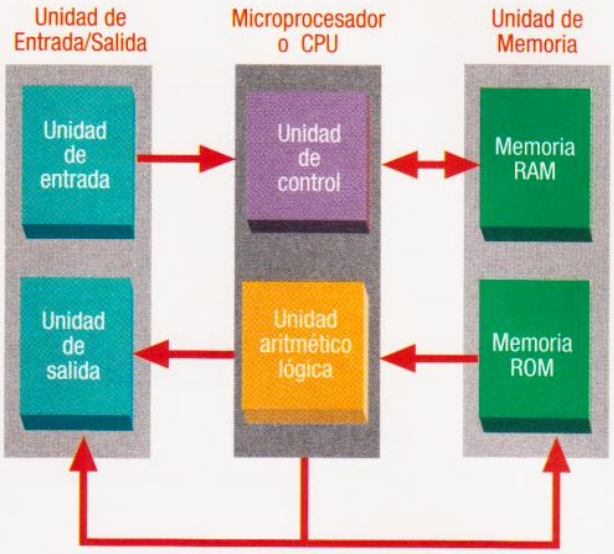


Figura 2.2 *Estructura general de una computadora digital*

La unidad de memoria se encarga de almacenar los datos y los programas que operan sobre esos datos y es una de las más importantes de una computadora. Se distinguen dos sistemas diferentes de memoria: la de almacenamiento primario y la de almacenamiento secundario. La memoria de almacenamiento primario se refiere a los circuitos que guardan los programas que se van a ejecutar y los datos que se necesitan durante la ejecución de estos programas, y la memoria de almacenamiento secundario, se utiliza para almacenar grandes cantidades de datos que no se requieran con frecuencia para la operación de la computadora. Sistemas de este tipo son los discos duros y los discos de 3.5".

Dentro de la memoria de almacenamiento secundario se distinguen tres categorías: la ROM (*Read Only Memory*) o memoria de solo lectura, donde se almacenan cierto tipo de programas como el del sistema de arranque de una computadora (BIOS); la RAM (*Random Access Memory*) o memoria de lectura y escritura, donde se alma-

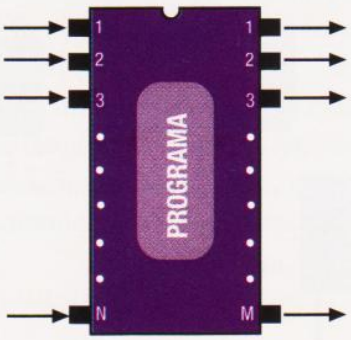
cenan los datos que los programas van generando y la cache que es una memoria RAM con tiempo de acceso muy rápido donde se mantienen los datos más utilizados durante la ejecución de un programa.

La unidad central de proceso (CPU: *Central Processing Unit*) reúne la unidad de control y la unidad aritmético-lógica en un solo bloque. Esta última también se conoce como ALU (*Arithmetic Logic Unit*). En la práctica, la unidad central o CPU se encuentra en forma de un circuito integrado llamado ***microprocesador***.

Dentro de esta CPU, la unidad de control se encarga de la interpretación y ejecución de las instrucciones del programa. También controla todos los componentes de una computadora por medio de líneas de conexión llamadas buses. La ALU se encarga de realizar las operaciones lógicas y aritméticas. Las principales funciones aritméticas realizadas en la ALU incluyen la AND, la OR, la EXOR y la comparación. Como hemos visto, el microprocesador es quien lleva a cabo las principales funciones de un sistema de cómputo, y quien integra a los demás componentes y ejecuta las instrucciones. Es por eso el principal componente de una computadora.

Funcionamiento del microprocesador

Desde un punto de vista conceptual, un microprocesador ideal es un dispositivo digital que acepta o lee datos aplicados a un cierto número de líneas de entrada (N); los procesa de acuerdo a las instrucciones secuenciales de un programa almacenado en su memoria, y suministra o escribe los resultados del proceso en un cierto número de líneas de salida (M), **figura 2.3**



El microprocesador (ideal) es un dispositivo con N líneas de entrada, M líneas de salida y un programa almacenado

Figura 2.3 *El microprocesador ideal*

Los datos de entrada pueden provenir de interruptores, sensores, convertidores A/D, teclados, etc. Los datos de salida pueden estar dirigidos a actuadores, indicadores o *displays*, pantallas, convertidores D/A, alarmas, impresoras, etc. El programa almacenado determina como deben ser procesados los datos de entrada y, en consecuencia, que información debe enviarse a las líneas de salida.

El soporte físico de las instrucciones del programa es la memoria, la cual almacena los datos para que sean procesados. En un momento dado, los niveles lógicos (unos y ceros) de las líneas de salida de un microprocesador, dependen no solamente del programa en sí sino también de la historia de las señales de entrada hasta ese momento. Mientras que en un microprocesador ideal no existen restricciones respecto al número de entradas y salidas, los microprocesadores reales sólo pueden acomodar un número limitado de terminales o pines para estas funciones. Como veremos más adelante, para simplificar el número de pines, la mayoría de microprocesadores utilizan las mismas líneas para la entrada y salida de datos.

De otro lado, mientras que un microprocesador ideal tiene, teóricamente, una memoria interna ilimitada, los microprocesadores reales solo disponen de una cantidad limitada para almacenar

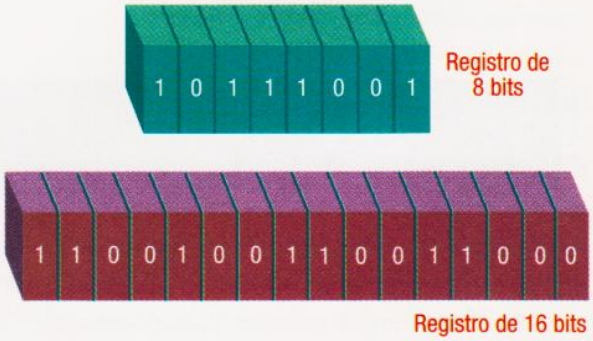


Figura 2.4 *Registros de un microprocesador*

datos e instrucciones. Por esta razón, un microprocesador debe tener la posibilidad de comunicarse con una memoria externa. Para lograrlo, necesita disponer también de un conjunto de líneas de selección o direccionamiento adicionales.

Finalmente, para sincronizar su operación con la de los componentes externos conectados al mismo y tener un control global de los buses de datos y direcciones, un microprocesador necesita disponer también de un conjunto de líneas de control. Por tanto, un microprocesador requiere de un bus de datos, un bus de direcciones y un bus de control para comunicarse con los demás componentes.

Naturalmente, el microprocesador ideal (un dispositivo con sólo entradas y salidas y un programa almacenado en él) no existe. Sin embargo, la tendencia de los nuevos diseños es incluir cada vez más funciones en una misma pastilla, incluyendo unidades de memoria e interfaces entrada/salida como en el caso de los microcontroladores.

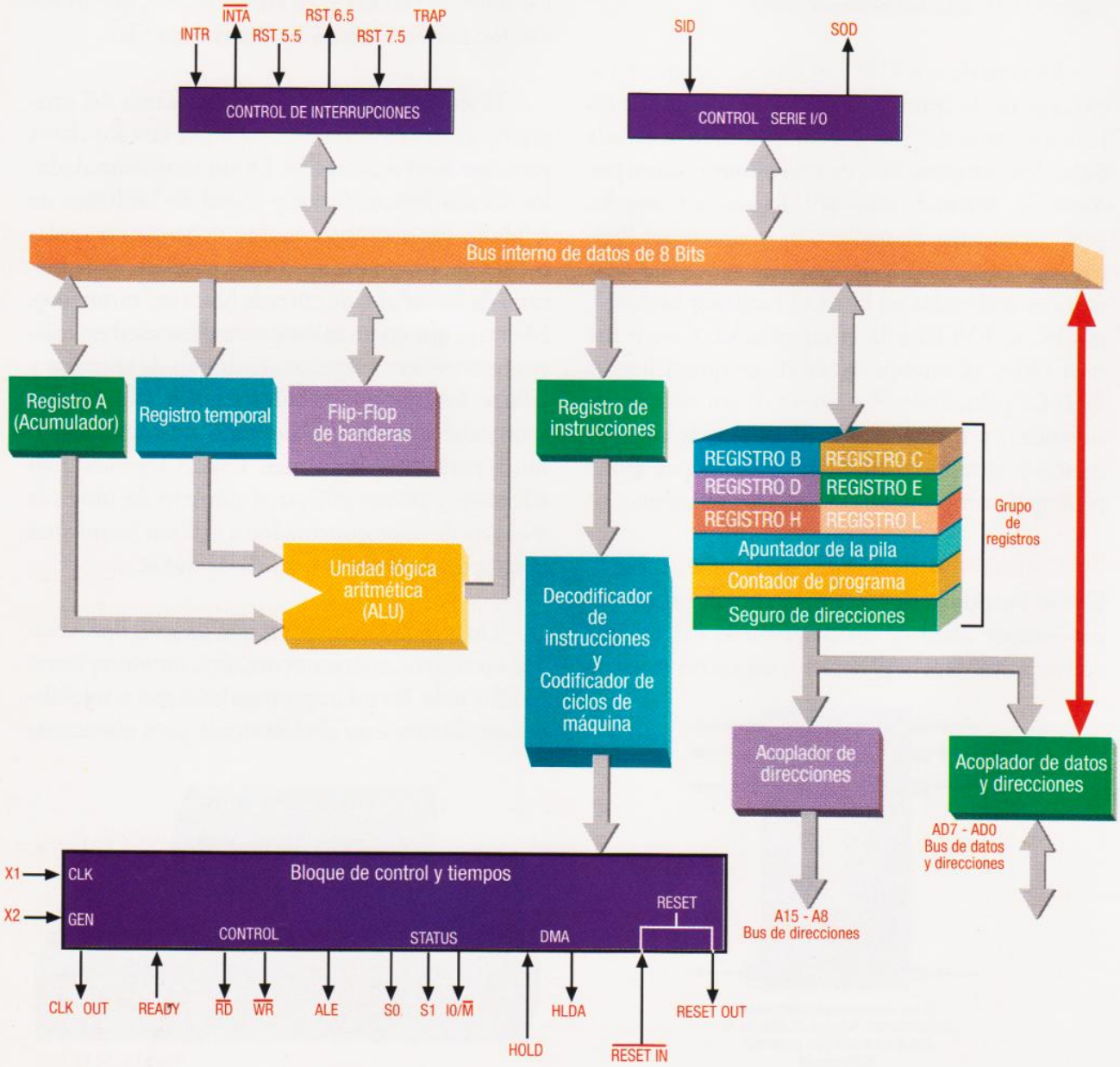


Figura 2.5 *Arquitectura interna de un microprocesador 8085 de Intel*

Organización interna de un microprocesador

La organización interna de un microprocesador, es decir, su arquitectura, varía notablemente de un dispositivo a otro. Por esta razón, es difícil definir un modelo de microprocesador que represente todas las alternativas posibles puesto que cada uno tiene una lógica de funcionamiento propia. La arquitectura, y especialmente el tipo, número y organización de los registros internos, es una consideración importante a la hora de elegir un microprocesador para una aplicación determinada.

Un registro es una posición de memoria conectada a la CPU donde se pueden retener cifras binarias. Está formado por elementos lógicos (compuertas, *flip-flops,* etc.) que, al ser tomados en conjunto, pueden almacenar números binarios de 4, 8, 16 ó más *bits*, **figura 2.4**. Se utilizan esencialmente para almacenamiento temporal, en el que el contenido cambia continuamente.

Esto no implica que se deba conocer con demasiado detalle, cómo funciona internamente un microprocesador para poder utilizarlo. Existen partes y funciones que normalmente deben estar presentes: un reloj, una ALU, varios registros, un contador de programa, etc. En la **figura 2.5** se muestra un ejemplo de la arquitectura interna de un microprocesador simple, el 8085 de Intel.

Por ejemplo, el reloj o circuito de oscilación puede estar o no incorporado en el microprocesador, pero es absolutamente necesario puesto que tiene la misión de sincronizar todas las operaciones de debe realizar el dispositivo. Del mismo modo, para ejecutar un programa, un microprocesador debe tener en su interior registros que le permitan almacenar información, realizar lecturas, modificaciones a los datos, etc.

Operación básica de un microprocesador

Analizando un sistema de cómputo, nos damos cuenta que el microprocesador o CPU es la parte del sistema encargada de leer, interpretar y ejecutar las instrucciones del programa almacenado en la memoria.

Desde un punto de vista conceptual, la acción de leer e interpretar las instrucciones la realiza una parte del microprocesador denominada unidad de instrucción y su ejecución corre a cargo de otro circuito especializado dentro del mismo denominado unidad de ejecución, **figura 2.6**

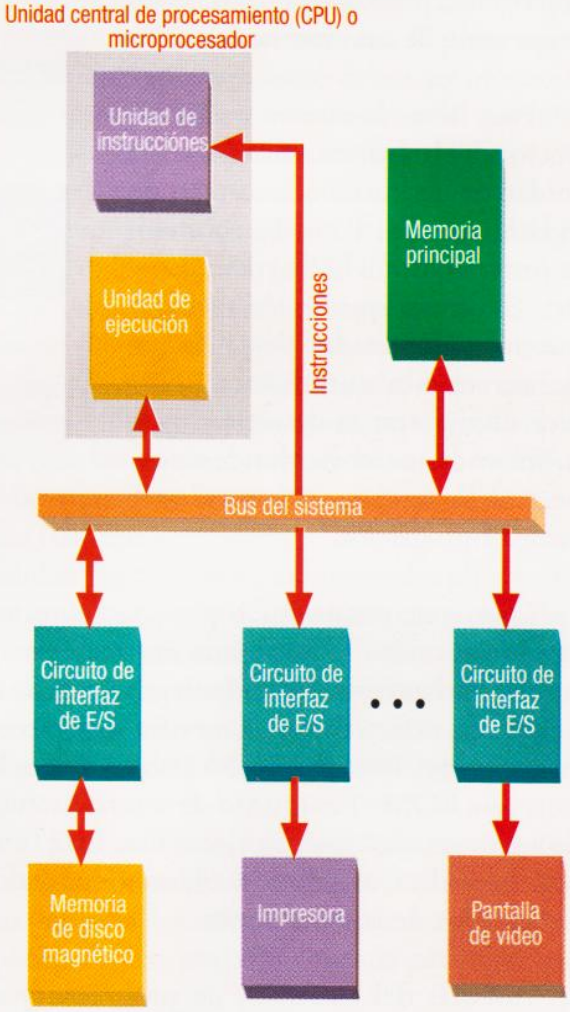


Figura 2.6 *Unidades de instrucción y ejecución*

La unidad de ejecución, a su vez, contiene una unidad aritmético-lógica o ALU y un conjunto de registros auxiliares. La ALU se encarga de realizar una gran variedad de operaciones aritméticas y lógicas, y los registros de almacenar temporalmente información. Las distintas funciones realizadas por el microprocesador quedan especificadas por su conjunto de instrucciones.

La unidad de instrucción lleva la cuenta de las posiciones o direcciones de memoria donde se encuentran las instrucciones que el microprocesador necesita en cada instante. Normalmente, las instrucciones se leen y ejecutan en el mismo orden en que fueron almacenadas en la memoria. Sin embargo, la presencia de instrucciones de control de programa puede alterar la secuencia de ejecución de estas instrucciones.

Para llevar la cuenta de las direcciones de las instrucciones, la unidad de instrucción incorpora un registro especial llamado contador de programa o PC (*Program Counter*). Cada vez que se lee una instrucción, el contador de programa se incrementa automáticamente para suministrar la dirección de la posición de memoria donde se encuentra la siguiente instrucción o dato del programa.

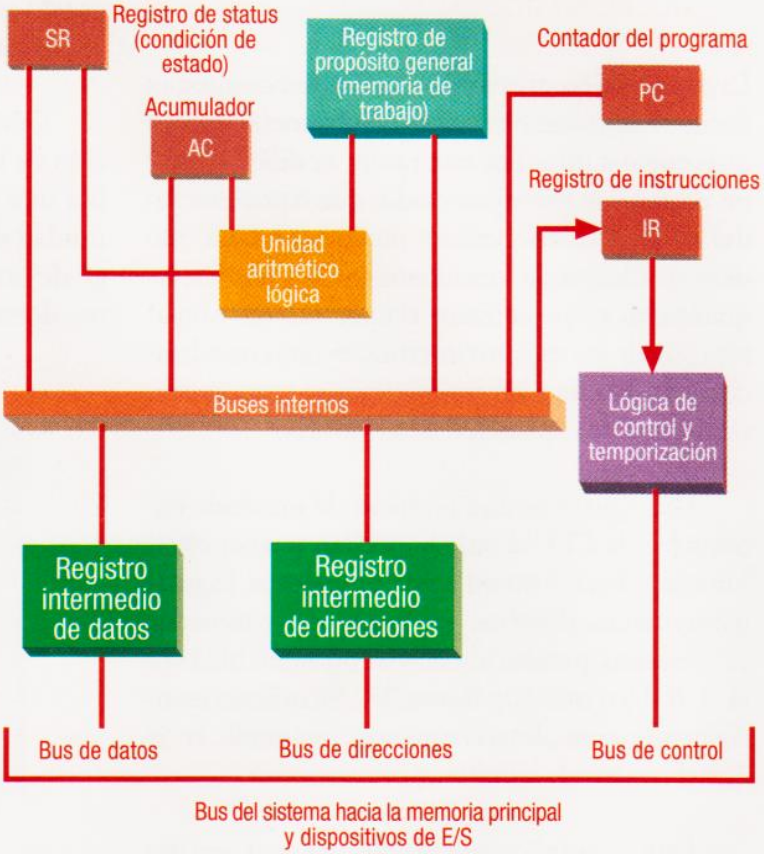


Figura 2.7 *Arquitectura de un microprocesador genérico*

Cada vez que un microprocesador recibe una instrucción, la almacena en un registro interno del mismo llamado registro de instrucciones con el fin de proceder a su decodificación o interpretación y buscar en la memoria ROM el conjunto de microinstrucciones necesarios para su ejecución. Esta función la realiza un circuito denominado decodificador de instrucciones.

Además del contador de programa y el registro de instrucciones, un microprocesador contiene usualmente otros registros para facilitar el acceso a las instrucciones y a los datos. Algunos de estos registros son programables en el sentido de que su contenido puede ser alterado por *software* (instrucciones) mientras que otros son inaccesibles desde el punto de vista de programación y su contenido sólo lo puede determinar y consultar el microprocesador.

La mayoría de los microprocesadores, por ejemplo, incluyen un registro especial llamado acumulador (Registro A) que es utilizado por muchas instrucciones como fuente o destino de datos. También es muy común la presencia de un registro de estado que suministra información relativa a la ejecución de ciertas instrucciones como sobreflujo, paridad, generación de acarreo, resultado negativo o cero, etc.

Muchos microprocesadores poseen también un conjunto de registros de propósito general que pueden ser utilizados para almacenar direcciones de memoria, datos, resultados intermedios y otros propósitos. El número y tipos de registros que posee un microprocesador, es una parte muy importante de su arquitectura y programación.

En la **figura 2.7** se muestra un modelo genérico de microprocesador que ilustra los conceptos anteriores.

Estructura de buses de un microprocesador

En la **figura 2.8** se muestra la estructura de buses generalizada de un microprocesador. Externamente, el dispositivo cuenta con un bus de datos, un bus de direcciones y un bus de control. En las siguientes secciones decribiremos la función y las características de cada uno. Además de estos tres buses principales, un microprocesador debe disponer también de un bus de alimentación, el cual hace llegar la corriente de la fuente a sus distintos componentes internos.

Figura 2.8 *Estructura de buses generalizada*

Las líneas de los buses de un microprocesador transportan voltajes que representan números binarios (1's y 0's). El microprocesador, los puertos de entrada/salia y la memoria, responden únicamente a esos números binarios codificados electrónicamente.

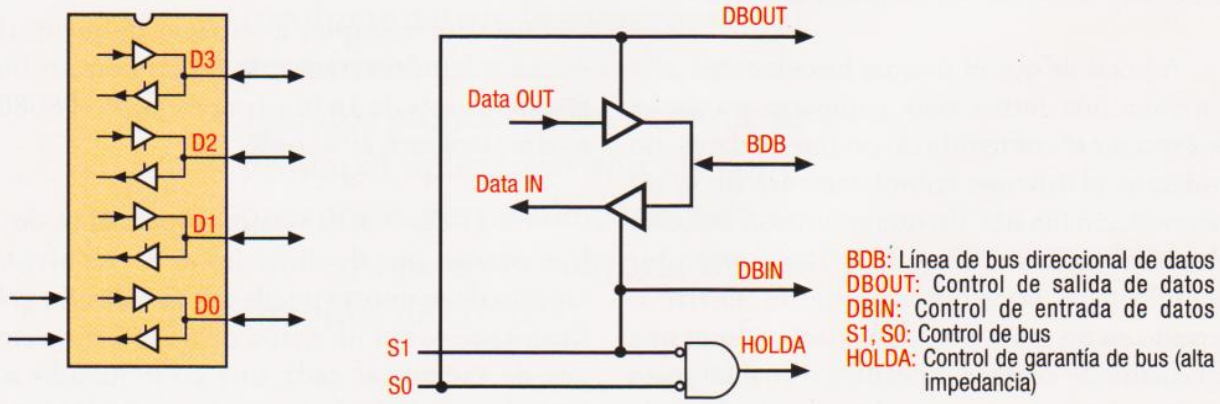
El bus de datos. Líneas bidireccionales

El bus de datos lleva datos e instrucciones hacia y desde el microprocesador. Las instrucciones proceden siempre de la memoria mientras que los datos que procesa u obtiene el programa de instrucciones, puede provenir de o ir hacia la memoria o los módulos de entrada/salida.

Generalmente, el número de líneas de entrada es igual al número de líneas de salida. Este número define la longitud de la palabra de datos del microprocesador. Son comunes longitudes de palabra de 4, 8, 16, 32 y 64 bits. En la **figura 2.4**, por ejemplo, se muestra un microprocesador de 8 bits. En este caso, tanto el bus de datos de entrada como el bus de datos de salida con de un *byte*, es decir de ocho (8) bits.

El número total de pines asignado en el circuito integrado del bus de datos puede reducirse prácticamente en un 50% si se utilizan los mismos puntos de conexión tanto para los datos de entrada como para los datos de salida, pero no para ambos simultáneamente. El bus de datos así constituido es bidireccional en el sentido de que la información puede circular entrando o saliendo del microprocesador. La forma de representar un bus de datos bidireccional para un microprocesador de N bits se ilustra en la **figura 2.9**

Un bus bidireccional puede construirse con compuertas *tri-state* para controlar la dirección del flujo de información. La **figura 2.9 (a)** representa un microprocesador de cuatro bits con un bus de datos bidireccional. En la figura **2.9 (b)** se muestra la configuración de una línea de bus bidireccional. Las líneas de selección S1 y S0 proceden del bus de control.



(a) (b)

Figura 2.9 *(a) Microprocesador de 4 bits con bus de datos bidireccional. (b) Línea de bus bidireccional.*

La transferencia de datos de entrada o salida desde o hacia el bus de datos se realiza, respectivamente, con S1S0 = 10 y S1S0= 01. La línea de bus puede inhabilitarse con S1S0 = 00 para impedir cualquier transferencia de información entre el bus de datos y el microprocesador.

Las líneas de selección pueden utilizarse para informar a los módulos externos que comparten el bus de datos el estado del bus bidireccional en un momento dado. Este es el propósito de las líneas marcadas como DBOUT (bus de datos en el modo de salida), DBIN (bus de datos en el modo de entrada) y HOLDA (bus de datos en estado de alta impedancia) en la **figura 2.9 (b)**.

La información contenida en el bus de datos puede representarse numéricamente de varias formas: binaria, octal y hexadecimal, principalmente. Considérese, por ejemplo, un microprocesador de 8 bits con el *byte* 10111011 en las líneas D7 a D0 del bus de datos, siendo D7 el bit más significativo o LSB y D0 el bit menos significativo o LSB.

Esta palabra de datos representada como un número binario es simplemente 10111011(B) ó 10111011(2), representada como número octal es 273Q ó 2738 y como número hexadecimal (la más usada) es BBH ó BB16. La forma de convertir números binarios en octales o hexadecimales y viceveresa se explicará en detalle en la sección de programación. Los sufijos B,Q y H, y los subíndices 2, 8 y 16 enfatizan el sistema numérico (binario, octal y hexadecimal) de cada presentación.

A pesar de que el sistema hexadecimal proporciona una forma muy compacta y concisa de expresar el contenido de un bus de datos, no podemos olvidarnos completamente de la representación binaria. En una aplicación real, esta última nos permite identificar fácilmente, por ejemplo, cual interruptor ha de ser abierto o cerrado en un momento dado para obtener una condición de entrada específica o cual lámpara ha de ser energizada para obtener un efecto de salida determinado.

El bus de direcciones. Espacio de memoria

El bus de direcciones contiene la información digital que envía el microprocesador a la memoria y demás elementos direccionables del sistema para seleccionar una posición de memoria, una unidad de entrada/salida o un registro particular de la misma. El número de líneas disponible en el bus de direcciones (n) determina el tamaño máximo de memoria que puede ser acomodado en el sistema (2^n).

Así, por ejemplo, un microprocesador con un bus de datos de 8 bits y un bus de direcciones de 16 bits (típico) tal como el Z-80 puede manejar directamente una memoria de 2^{16} = 65536 (64K) posiciones, cada una conteniendo un *byte* (palabra de 8 bits).

Naturalmente, una vez seleccionada una posición de memoria, el microprocesador debe estar en capacidad tanto de almacenar información en esa localización (operación de escritura) como de extraerla (operación de lectura).

La cantidad de memoria utilizada en un sistema con microprocesador depende de la aplicación específica y es a menudo inferior a la máxima que puede manejar el bus de direcciones. El conjunto completo de localizaciones de memoria a las que puede tener acceso un microprocesador, se denomina espacio de memoria.

Así, por ejemplo, el espacio de memoria de un microprocesador de 8 bits con un bus de direcciones de 16 bits (por ejemplo el 8080) es simplemente 2^{16} = 64K.

En el caso de un microprocesador de 8 bits con un bus de direcciones de 16 bits, el espacio de memoria puede ser visualizado gráficamente como un mapa de 65536 direcciones de memoria, cada una conteniendo un byte, la más baja de las cuales es 0000H y la más alta FFFFH, **figura 2.10**

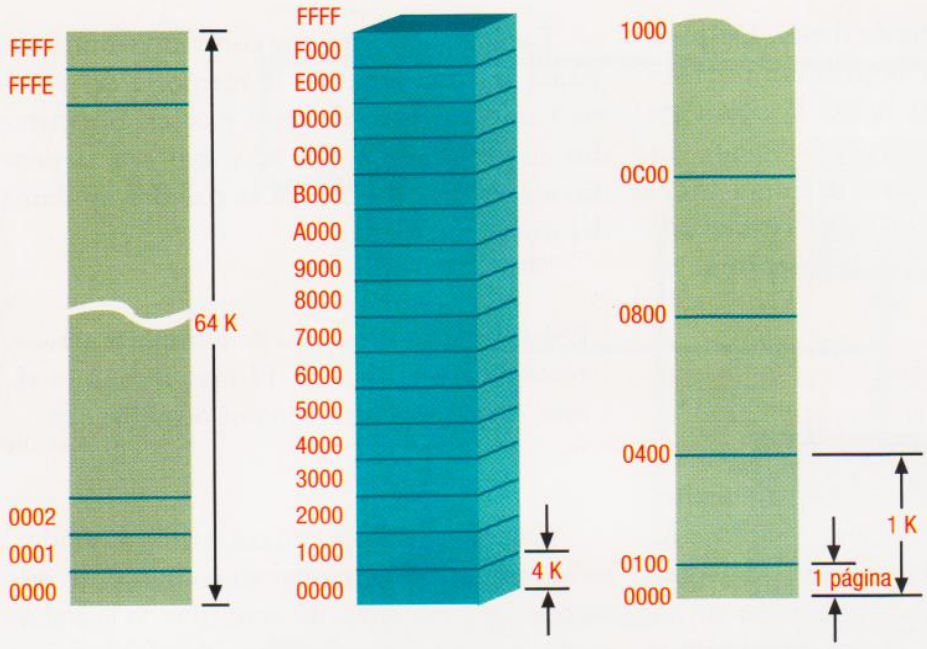


Figura 2.10 Espacio de memoria

Note que la dirección de memoria se incrementa en 1000H cada 4K (=4096) direcciones de espacio de memoria. Observe también que cada 1K (=1024) posiciones de memoria, la dirección de memoria se incrementa en 400H. La memoria puede dividirse también en páginas de memoria, cada una contiene 256 palabras y cubre 100H direcciones de memoria. Hay, por tanto, 4 páginas en 1K de memoria. Las direcciones en la página de más bajo orden van desde 0000H hasta 00FFH, en la siguiente desde 0100H hasta 01FFH, y así sucesivamente.

El bus de control

El bus de control contiene la información que envía el microprocesador a los elementos del sistema o bien la recibe de estos con el propósito de sincronizar la operación de los circuitos externos. El número de líneas del bus de control es variable y depende del microprocesador particular utilizado.

Organización de un sistema basado en microprocesador

En la **figura 2.11** se muestra la estructura organizativa de un sistema típico basado en microprocesador. Note que consta de cuatro elementos principales: una memoria, un microprocesador, un puerto de entrada y un puerto de salida.

El microprocesador está conectado a todos los otros componentes a través del ***Bus de datos***. Las señales del ***Bus de control***, conjuntamente con las del ***Bus de direcciones***, determinan que elementos se comunican con el microprocesador en un momento dado. Estos tres buses configuran el bus del sistema y el gobierno del mismo es responsabilidad del microprocesador.

La secuencia de instrucciones que constituyen el programa que debe ejecutar el microprocesador están almacenadas en un área de la memoria. En el momento de iniciar el sistema, el microprocesador sitúa en el bus de direcciones la dirección de la posición de memoria donde se encuentra la primera instrucción. Como resultado, la memoria entrega

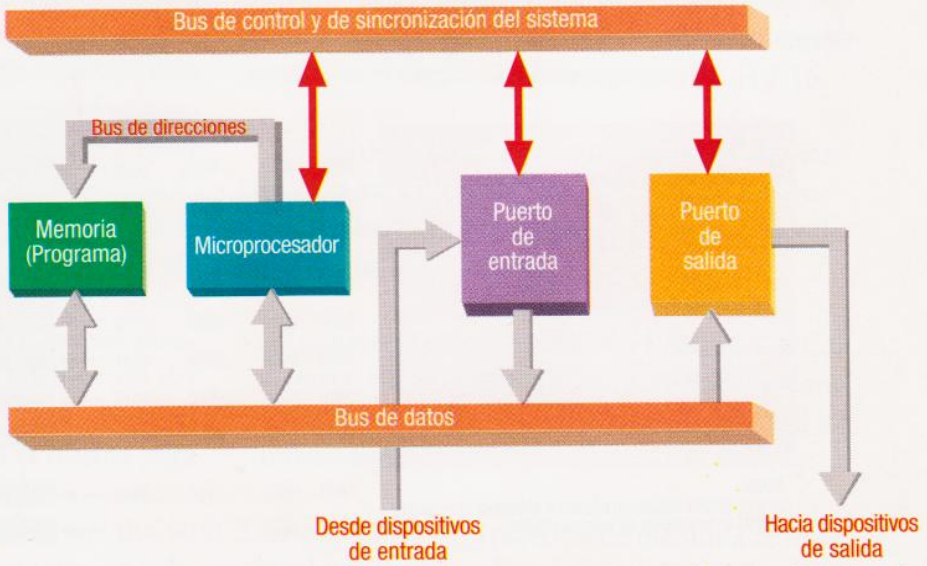


Figura 2.11 Diagrama de bloques de un sistema basado en un microprocesador

esta información en el bus de datos. Después de interpretar y ejecutar la primera instrucción, el microprocesador busca la siguiente instrucción, la ejecuta y así sucesivamente. La generación de la serie de instrucciones necesarias para que el microprocesador realice una tarea determinada, es lo que se denomina su programación.

Secuencia de operación de un sistema basado en un microprocesador

Hasta aquí hemos visto la arquitectura genérica de un microprocesador y de un sistema basado en uno de estos dispositivos; ahora analizaremos con más detalle los pasos que sigue un sistema de esta naturaleza para ejecutar las instrucciones contenidas en un programa y aprenderemos como trabajan en conjunto cada uno de los bloques del sistema para desarrollar estas tareas.

Un microprocesador es un diminuto conjunto de miles o millones de elementos lógicos e interconexiones diseñado para responder a las instrucciones de un *programa*, el cual le indica lo que debe hacer en cada instante. Sin un programa que le de vida, un microprocesador y todo el *hardware* conectado a su alrededor, sería una masa inerte de componentes electrónicos sin un objetivo específico. De hecho, la función primaria de un microprocesador es ejecutar programas.

Las instrucciones que constituyen un programa se almacenan en la memoria del sistema como patrones de unos y ceros, organizados en grupos de 8, 16, 32 y más *bits*, dependiendo de la longitud de la palabra de datos del microprocesador.

> PALABRA: Es un grupo de *bits* que se tratan como una sola unidad. El más común es el *byte*, el cual está formado por ocho (8) *bits*.

Cuando se trabaja con un microprocesador, es importante comprender su conjunto de instrucciones y ser capaz de utilizarlas en la elaboración de programas; se debe entender la forma como el microprocesador las ejecuta y adquirir la habilidad para emplearlas eficientemente en la manipulación de datos.

Para ejecutar las operaciones indicadas por una instrucción, el microprocesador debe empezar por direccionar o encontrar la posición de memoria donde se encuentra la instrucción, leerla y decodificarla o interpretarla por medio de sus circuitos internos. Los pasos anteriores constituyen lo que se denomina ***ciclo de instrucción***. Durante su operación normal, un microprocesador no hace otra cosa que ejecutar repetidamente ciclos de instrucción.

Direcciones (En hexadecimal)	Instrucciónes, dato (En binario)
0 0 0 0 H	1 0 1 1 1 1 0 1
0 0 0 1 H	0 0 0 0 0 0 0 1
0 0 0 2 H	1 1 1 0 0 1 1 0
0 0 0 3 H	0 0 0 0 1 1 1 1
0 0 0 4 H	0 0 1 1 0 0 1 0
0 0 0 5 H	0 1 0 0 0 0 0 0
0 0 0 6 H	0 0 1 0 0 0 0 0
0 0 0 7 H	0 1 1 1 0 1 1 0

NOTA:
Este programa calcula un número y almacena el resultado (DCBA) en la posición de memoria 2040H.

Direcciones	Instrucciones, datos
0000 0000 0000 0000	1 0 1 1 1 1 0 1
0000 0000 0000 0001	0 0 0 0 0 0 0 1
0000 0000 0000 0010	1 1 1 0 0 1 1 0
0000 0000 0000 0011	0 0 0 0 1 1 1 1
0000 0000 0000 0100	0 0 1 1 0 0 1 0
0000 0000 0000 0101	0 1 0 0 0 0 0 0
0000 0000 0000 0110	0 0 1 0 0 0 0 0
0000 0000 0000 0011	0 1 1 1 0 1 1 0
0010 0000 0100 0000	0 0 0 0 D C B A
1111 1111 1111 1111	X X X X X X X X

Figura 2.12 *Ejemplo de un programa en lenguaje de máquina*

Direcciones (En hexadecimal)	Contenido Instrucción, dato (Hex)
0 0 0 0	BD
0 0 0 1	01
0 0 0 2	E6
0 0 0 3	0F
0 0 0 4	32
0 0 0 5	40
0 0 0 6	20
0 0 0 7	76

Figura 2.13 *Ejemplo de un programa objeto*

Para ejecutar un programa, un microprocesador debe tenerlo almacenado en forma binaria (patrones de **1** y **0**) en posiciones adyacentes de la memoria de programa del sistema. Esta forma binaria del programa se denomina *lenguaje de máquina* y es el único lenguaje que entiende el microprocesador. En la **figura 2.12** se muestra un ejemplo de un programa en lenguaje de máquina para un microprocesador 8085. Para evitar la confusión que, desde el punto de vista del programador, puede generar una larga lista de unos y ceros, estos códigos se pueden representar en forma hexadecimal, constituyendo lo que se denomina un *programa o código objeto*. En otras palabras, un código objeto es, simplemente, una representación abreviada de un programa en lenguaje de máquina, **figura 2.13**. Por ejemplo, **101111 01** se representa como **BD**, **0000 0001** como **01**, **1110 0110** como **E6**, **0000 1111**como **0F**, etc.

No todos los códigos de un programa objeto o en lenguaje de máquina, corresponden a instrucciones ejecutables. Muchos de ellos corresponden a datos y direcciones que son utilizados por el programa para propósitos particulares. Los códigos correspondientes a instrucciones ejecutables se denominan comúnmente códigos operacionales (*opcodes*). Por tanto, en un sistema de 8 *bits*, los programas en lenguaje de máquina se reducen a unas cuantas secuencias de grupos de 8 *bits* (*1 byte*) situados en algún lugar de la memoria.

Estas secuencias representan a su vez una mezcla de instrucciones para el microprocesador (*opcodes*) y datos sobre los cuales ha de operar el mismo. Cada código de operación le indica, directa o indirectamente al microprocesador, la dirección de memoria donde se encuentra almacenada la siguiente instrucción ejecutable. Al recibir un código de operación, los circuitos internos del microprocesador lo descifran. La información obtenida permite al dispositivo identificar la naturaleza de los *bytes* que siguen al *opcode*.

El conjunto de códigos de operación (*opcodes*) a los cuales puede responder un microprocesador, constituye su conjunto de instrucciones *(Instruction Set)*. Cada microprocesador tiene su propio conjunto de instrucciones, cada una representada por un *opcode* diferente. El número máximo de códigos de operación que puede descifrar un microprocesador es igual a 2^N, siendo N el tamaño de la palabra de datos del mismo.

Las instrucciones de un microprocesador pueden agruparse en tres grandes categorías según su función: la primera son las instrucciones de transferencia de datos, las cuales mueven información sin alterar su contenido de una parte a otra del sistema; la segunda categoría es la de procesamiento de datos; estas instrucciones transforman la información desde el punto de vista lógico o aritmético; y la tercer categoría, son las instrucciones de control de programa, ellas determinan la secuencia de ejecución de las instrucciones.

En la **figura 2.14**, observamos un diagrama simplificado de la arquitectura de un sistema basado en un microprocesador, en el cual basaremos nuestro an álisis de la interacción entre cada una de las partes del sistema y la secuencia que presenta.

En este diagrama se pueden notar cuatro bloques fundamentales: la memoria, la unidad de control, la unidad operativa y el módulo de entradas y salidas. Todos ellos conectados a través de los *buses* de direcciones, datos y control.

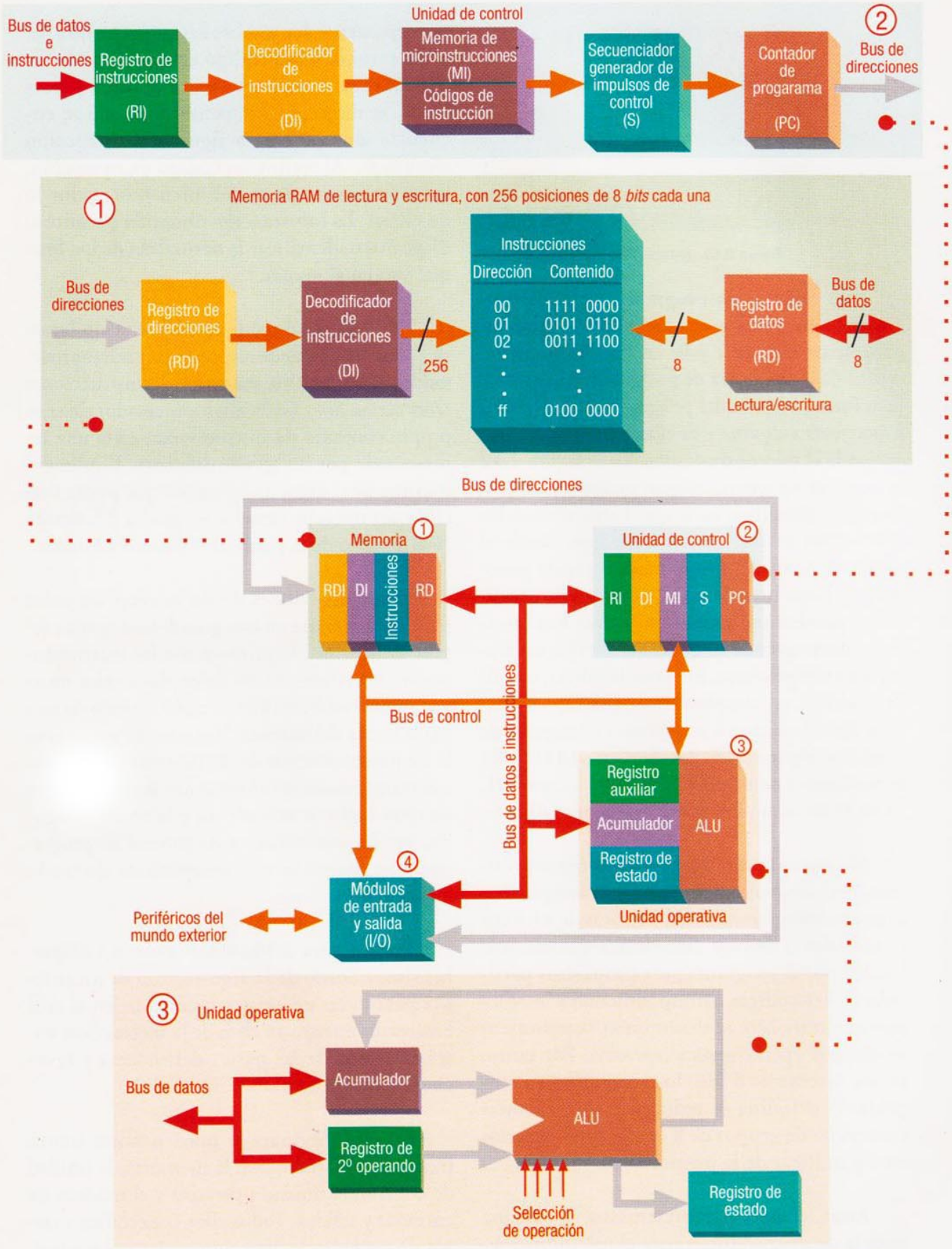


Figura 2.14 Diagrama de bloques simplificado de un sistema basado en un microprocesador

El **bloque 1** es la ***memoria***; en ella se almacenan una serie de cantidades codificadas en ocho 8 *bits*. Para entender la operación de una memoria, puede hacerse una analogía con un casillero de correos, donde existen muchas casillas para diferentes usuarios, cada una con un número o dirección que la identifica y con la propiedad de almacenar cualquier clase de información.

Pues bien, una memoria posee un conjunto de posiciones cada una con su propia dirección, y en cada una de dichas posiciones puede existir cualquier dato de ocho *bits*.

La posición que se va a acceder es indicada por el *bus* de direcciones al ***Decodificador de Direcciones*** (***DI***), el cual es un dispositivo de n entradas y 2^n salidas. Dependiendo de la entrada binaria que lea, habilita la lectura de una dirección determinada por medio de la activación de la línea correspondiente, **figura 2.15**. Los 8 *bits* de la posición accedida salen por el *bus* de datos hasta el ***Registro de Datos*** (***RD***), si se efectúa una operación de lectura.

Cuando se trata de una operacón de escritura, el contenido digital de las 8 líneas del *bus* de datos se carga en el Registro de Datos y, desde él, en las 8 celdas de la posición direccionada. El proceso de direccionar una posición y leer o grabar un dato se denomina ***ciclo de memoria***.

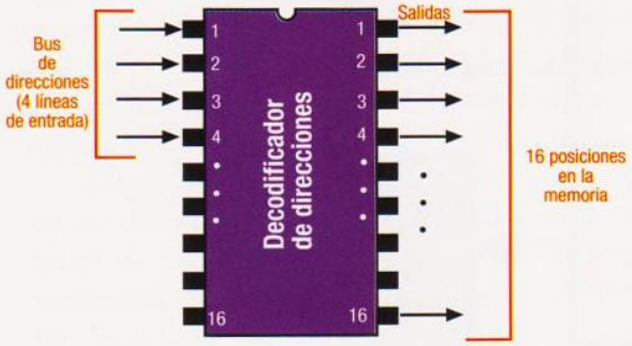


Figura 2.15 *Según el dato introducido por el bus de direcciones, se habilita la comunicación con una determinada posición de memoria*

RECUERDE QUE

La memoria principal de un sistema basado en un microprocesador se construye con memorias de tipo electrónico, que se clasifican en dos grandes grupos:

Memorias ROM (***Read Only Memory***). Son memorias que una vez grabadas, permanecen con dicha información permanentemente y sólo pueden ser leídas. Son memorias de "sólo lectura" y se utilizan para guardar los programas o datos fijos.

Memorias RAM (***Random Access Memory***). El contenido de sus posiciones puede ser leído y escrito. Al igual que las memorias ROM, son de acceso aleatorio, es decir, que para acceder a una posición determinada no hay que pasar por las anteriores, como sucede en las cintas y discos magnéticos. Son las más rápidas y su tiempo de acceso se mide en nanosegundos.

El principal inconveniente de las memorias tipo RAM es que son volátiles, o sea, al cortarse el suministro de energía eléctrica, cuando se desconecta el sistema, pierden la información que contenían. En un sistema con microprocesador, se utilizan este tipo de memorias para almacenar el resultado de las operaciones y los datos de las variables que se van creando en el programa.

El **bloque 2** es la ***unidad de control*** cuya función primordial es la de interpretar y ejecutar las instrucciones. El código binario de la operación que se va a ejecutar, se deposita inicialmente en el ***Registro de Instrucciones*** (***RI***) y luego se traslada al ***Decodificador de Instrucciones*** (***DI***), cuya misión es seleccionar en una memoria ROM un conjunto de posiciones que corresponden al código recibido y en las que se encuentran los códigos de las operaciones elementales (o *microinstrucciones*) que componen las diferentes etapas en las que se divide la ejecución de la instrucción decodificada.

Las microinstrucciones van pasando al ***Secuenciador*** (***S***), que es el circuito lógico de con-

Figura 2.16 *Movimientos de información de los contenidos de las unidades que participan en la fase de búsqueda de una instrucción*

trol y tiempos el cual gobierna a todos los elementos del sistema y lleva a cabo la ejecución secuencial de las microinstrucciones. La función del ***Contador de Programa*** (***PC***) es enviar por el *bus* de direcciones la posición de la memoria donde se encuentra la siguiente instrucción que se va a ejecutar. Normalmente, este contador se incrementa en una unidad en cuanto la memoria acepta la dirección de la instrucción anterior.

El **bloque 3** es la ***unidad operativa***; allí se ejecutan las operaciones aritméticas, lógicas, de desplazamiento, de rotación, de incremento, etc. Uno de los operandos que interviene en la operación que va a efectuar la ALU (Unidad lógico-aritmética), procede de un registro de 8 *bits* llamado ***Acumulador***. El otro operando llega desde cualquier parte del sistema y se carga o almacena en un registro auxiliar. Un registro especial, denominado ***Registro de Estado***, contiene una serie de *bits* que actúan como señalizadores de alguna característica especial que se haya producido en la última operación efectuada por la ALU. Por ejemplo, un señalizador denominado Z (cero), pasa a valer uno (1) si el resultado de la operación que ha hecho la ALU ha sido cero (0).

El **bloque 4** es el de ***entradas y salidas*** y se encarga de suministrar al sistema los datos procedentes del exterior, así como de llevar los resultados a los periféricos del mundo real.

Secuencia de operación de una instrucción

En un microprocesador, el principio de ejecución de cualquier instrucción tiene carac-

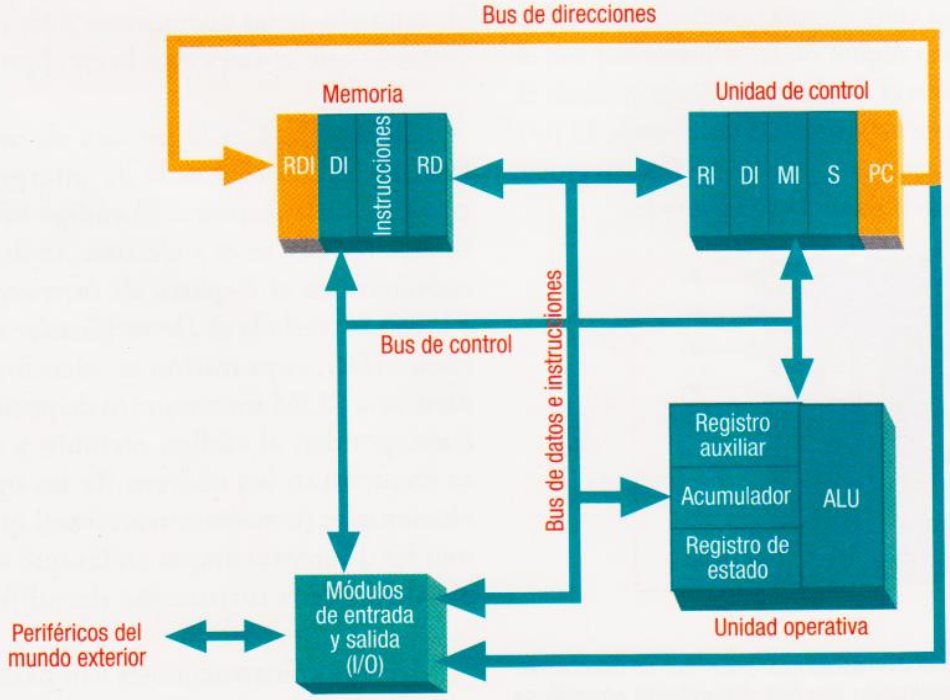


Figura 2.17 *El contenido del Contador de Programa (PC) pasa al Registro de Direcciones (RDI)*

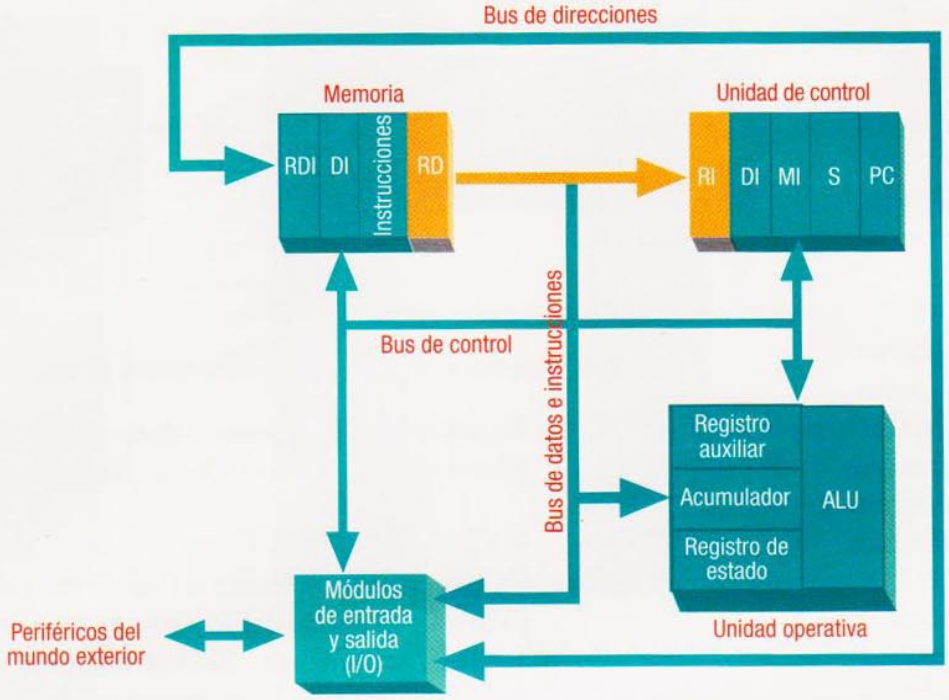


Figura 2.18 *El contenido de la posición de la memoria se deposita en el Registro de Instrucción*

terísticas similares. En todas ellas hay dos tiempos fundamentales llamados ***búsqueda*** (*fetch*) y ***ejecución*** (*execute*). El diagrama de bloques de un sistema con microprocesador permite un análisis rápido de la forma en que se ejecuta una instrucción cualquiera; por ejemplo, la correspondiente a la suma de dos operandos.

El contador de programa empieza conteniendo la dirección de la memoria principal donde está almacenado el código binario de la instrucción de suma. Luego ese contenido pasa a través del *bus* de direcciones hasta la memoria principal, donde se decodifica y selecciona la posición que contiene el código binario de la operación. Dicho código de "máquina" sale de la memoria por el *bus* de datos hasta el registro de instrucciones de la ***Unidad de Control***, donde se deposita. Toda esa fase de localización del código de la instrucción mencionada, recibe el nombre de ***fase de búsqueda*** y es la misma para cualquier instrucción.

El tiempo de ejecución empieza por un subtiempo de decodificación de la instrucción y prosigue con la parte de la ejecución propiamente dicha que puede tener varios subtiempos, dependiendo del tipo de instrucción.

Fase de búsqueda

Es el comienzo de una nueva instrucción, el ***Contador de Programa*** (*PC*) deposita en el *bus* de direcciones la dirección de la posición de la memoria principal donde se encuentra el código de la instrucción que se va a ejecutar. Dicho código sale de la memoria principal por el *bus* de datos hasta la ***Unidad de Control***, donde se graba en el ***Registro de Instrucciones***, **figura 2.16**.

Como la CPU no sabe que tipo de instrucción va a ejecutar, los tiempos de búsqueda son exactamente iguales. El desarrollo de esta fase es la siguiente:

1. El contenido del Contador de Programa (PC), a través del bus interno de la unidad de control, aparece en el registro de direcciones de memoria, **figura 2.17**.

2. El contenido de la posición de memoria aparece en el bus de datos. Esta información se deposita en un registro de instrucción (RI), **figura 2.18**.

Figura 2.19 *Comportamiento de la unidad de control durante la fase de ejecución*

Al mismo tiempo, el Contador de Programa (PC) se incrementa, puesto que él debe siempre señalar la dirección de la siguiente instrucción que se va a ejecutar, para cuando se necesite. De esta forma se ha completado la operación de *búsqueda* de una nueva instrucción.

Decodificación de una instrucción

Es la primera parte del tiempo de ejecución y es también exactamente igual para todas las instrucciones, puesto que aún no se sabe que operación hay que efectuar.

De momento, la parte de la instrucción llamada ***código de operación*** (*opcode*), se transfiere al Decodificador de Instrucciones (DI) y aquí empieza la parte diferente de la instrucción.

Al recibir el Decodificador de Instrucciones el código de la instrucción en curso, se encarga de seleccionar en la memoria de microinstrucciones aquellas que corresponden a dicho código. La llegada de las microinstrucciones al Secuenciador, origina una serie de señales de control que regulan la ejecución de las diferentes etapas en las que se descompone la instrucción.

En el caso de una suma, uno de los sumandos ha de estar contenido previamente en el ***Acumulador***, mientras que el otro llegará del registro auxiliar, generalmente, desde la memoria de datos, cuya dirección correspondiente vendrá acompañando al código de la operación en la instrucción.

La ALU efectuará la suma y el resultado se depositará en el Acumulador, al mismo tiempo que los *bits* señalizadores del Registro de Estado, tomarán el valor correspondiente en función del resultado.

Ejecución de una instrucción

En resumen: completada la fase de búsqueda, el código de máquina del ***Registro de Instrucciones*** llega al ***Decodificador de Instrucciones***, que se encarga de localizar las posiciones de la memoria de microinstrucciones que correspondan. Dichas microinstrucciones van introduciéndose al secuenciador a medida que se realiza el proceso. El secuenciador, con cada microinstrucción, envía una serie de señales de control a los elementos del sistema que deben actuar en cada momento. Ejecutadas todas las microinstrucciones que componen la instrucción, el Contador de Programa se incrementa en una unidad y el sistema pasa a la ejecución de la siguiente instrucción del programa, **figura 2.19**.

Hay varios tipos de instrucciones básicas que pueden agruparse de acuerdo con diferentes criterios, tales como funcionalidad, número de ciclos de memoria utilizados, número de palabras que utiliza, etc. Dependiendo de las características de la computadora y de su microprocesador, los diferentes tipos de instrucciones serán de una, dos o más palabras y necesitarán uno o más ciclos de memoria para llegar a obtenerse enteramente en la unidad de control. En función del número de palabras y del tipo de instrucción, la ejecución tiene características diferentes. Estos parámetros determinan, por ejemplo, el número de veces que el sistema debe acceder a la memoria para leer toda la instrucción.

Puertos de entrada y de salida

El concepto de entrada/salida abarca toda transferencia o intercambio de información (*datos*) entre un microprocesador (CPU) y un dispositivo externo o periférico específico. Cuando los datos fluyen hacia el microprocesador, se tiene el caso de una operación de ***entrada*** y cuando lo hacen en dirección al mundo externo, se tiene el caso de una operación de ***salida***.

Un paso importante en el diseño de cualquier sistema basado en microprocesadores es seleccionar los puertos de entrada/salida o circuitos de interfaz cuyo costo y rendimiento sean los más adecuados para una aplicación dada. La complejidad de un puerto I/O puede variar desde un sencillo *flip-flop* hasta un circuito integrado altamente especializado; por ejemplo, un controlador para un tubo de rayos catódicos o un *display* gráfico tipo LCD.

En general, las funciones que debe realizar un puerto o dispositivo de interfaz de entrada/salida se pueden resumir en los siguientes términos:

- Identificar direcciones con el fin de establecer la conexión con los buses de datos y de control del sistema cuando se selecciona un dispositivo periférico de entrada/salida específico.
- Interpretar órdenes. Generalmente, las órdenes enviadas por el microprocesador al puerto I/O se reducen a señales de lectura y escritura. Estas señales pueden venir ya decodificadas o necesitar una decodificación previa.
- Adaptar físicamente el sistema a los requisitos del periférico. Esto incluye la utilización de manejadores (*drivers*) de líneas de transmisión, eliminadores de ruido, optoacopladores, etc.
- Controlar los tiempos para la transferencia de información con el fin de garantizar que el flujo de datos entre el microprocesador y el periférico se haga de una manera ordenada y eficiente.

Un puerto o interfaz de entrada/salida puede construirse utilizando componentes normales de pequeña y mediana escala como compuertas, *flip-flops*, registros, codificadores, decodificadores, etc; o circuitos integrados de gran escala LSI (*Large Scale Integration*) especializados como la interfaz periférica programable (PPI) **8255** o el controlador programable de interrupciones (CPI) **8259**.

En la **figura 2.20** se muestra un ejemplo sencillo de un puerto de salida con un *flip-flop* tipo D cuya función es transferir el estado del *bit* de datos D0 a un diodo LED que actúa como dispositivo periférico. Esto sucede cuando en la línea de direccionamiento A15 hay un nivel **alto** y la señal de escritura ($\overline{WR}$) es **baja**. Después de deshabilitado, el *flip-flop* retiene la información previamente almacenada mientras el microprocesador se dedica a otras tareas.

En la **figura 2.21** se muestra un ejemplo sencillo de puerto de entrada con un *flip-flop*. En este caso, el dispositivo periférico es un interruptor y su estado (**alto** o **bajo**), se transfiere a la línea de datos D0 cuando en la línea de direccionamiento A15 hay un nivel **alto** y la línea de control de escritura ($\overline{RD}$) es **baja**. Una vez que el microprocesador acepta el *bit* D0 presente en el bus de datos, lo transfiere al acumulador y sitúa la salida del puerto de entrada en estado de alta impedancia con el fin de que no interfiera con otras transferencias de datos.

Puertos I/O programables. Una manera más elegante y eficiente de controlar el tránsito de informa-

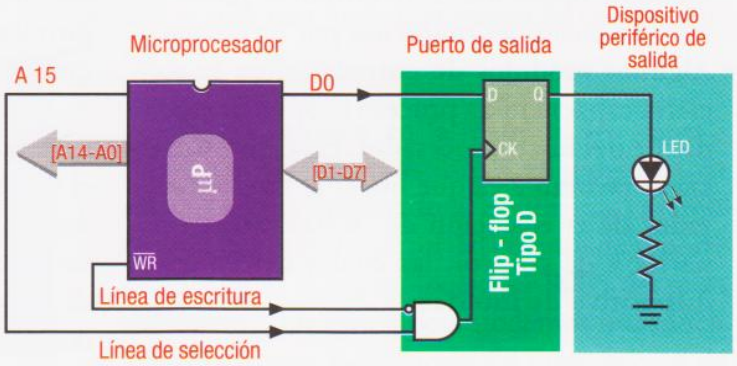


Figura 2.20 *Puerto de salida con flip-flop*

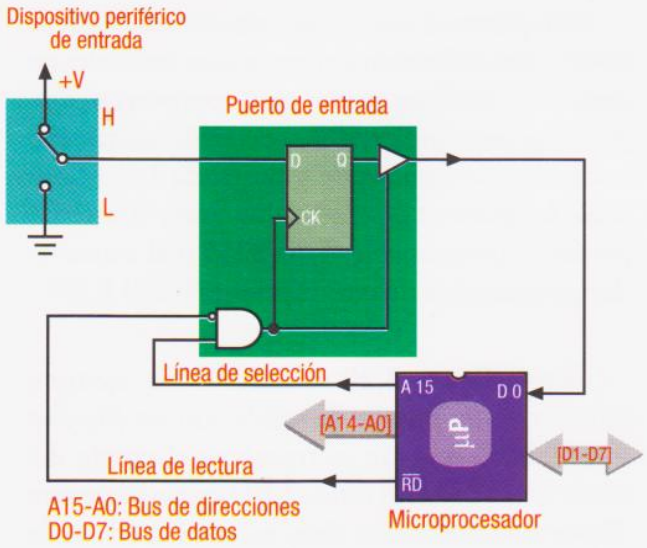


Figura 2.21 *Puerto de entrada con interruptor*

ción entre el mundo externo y el microprocesador, es utilizando circuitos integrados de gran escala (*LSI*) diseñados especialmente para el manejo de entradas y salidas. Entre las principales características de estos circuitos especializados podemos mencionar:

- Son fáciles de conectar con los buses del sistema puesto que disponen de líneas de adaptación directa con las salidas del respectivo microprocesador. Por lo general, sólo se requiere un sistema decodificador de direcciones.
- Son circuitos de entradas y salidas programables. Esta característica permite adaptarlos fácilmente a las condiciones de los circuitos externos utilizando únicamente comandos de programación. Generalmente, la programación consiste en cargar, de una forma predeterminada, un registro interno que define el modo de operación. Este último se puede alterar en cualquier momento por programa.
- Algunos módulos de entrada/salida poseen cierta capacidad de proceso propia, independiente del microprocesador. Esta circunstancia es favorable ya que libera al microprocesador de la ejecución de ciertas tareas rutinarias y de control, las cuales delega en el dispositivo de interfaz.

La característica anterior, denominada algunas veces *inteligencia ditribuida*, simplifica el circuito y el programa y hace posible funciones que no serían ejecutables si el microprocesador tuviera que hacerse cargo de todo el proceso de entrada/salida. Por la función que hacen, los puertos de interfaz programables se pueden dividir en dos categorías: interfaces dedicadas e interfaces de propósito general o interfaces universales.

Las ***interfaces dedicadas*** se especializan en una función específica, por ejemplo, controlar periféricos, temporizar eventos, etc. En esta categoría se encuentran entre otros: el temporizador programable **8253**, el controlador de disco flexible **8271**, la interfaz programable de teclado y *display* **8279** y el controlador de tubos de rayos catódicos (TRC) **8275**.

Las ***interfaces de propósito general***, como su nombre lo indica, son más universales y pueden adaptarse a una gran variedad de aplicaciones. Existen interfaces para transmitir datos en paralelo como la interfaz periférica programable (PPI) **8255** y para la transmisión de datos en serie como el **8251**.

El circuito de soporte 8155 (I/O-RAM)

Si analizamos un microprocesador típico como el 8085, observaremos que contiene todas las partes de una CPU funcional así como un puerto serie; todo lo que necesita para completar un sistema básico es una memoria RAM, una memoria ROM y puertos paralelos de entrada/salida.

Con el fin de minimizar el número de circuitos integrados necesarios para configurar un sistema sencillo, Intel, creador del **8085**, ha desarrollado una serie de componentes que integran memoria RAM y puertos I/O ó memoria ROM y puertos I/O en un mismo circuito integrado. Los dispositivos más representativos son el **8155**, el **8355** y el **8755**. El **8155** contiene 256 *bytes* de RAM, dos puertos I/O paralelos de 8 *bits*, un puerto I/O paralelo de 6 *bits* y un temporizador de 14 *bits*. El **8355** contiene 2048 *bytes* de ROM y dos puertos I/O paralelos de 8 *bits*. El **8755** tiene dos puertos I/O y 2048 *bytes* de EPROM. En esta sección estudiaremos un sistema basado en micropro-

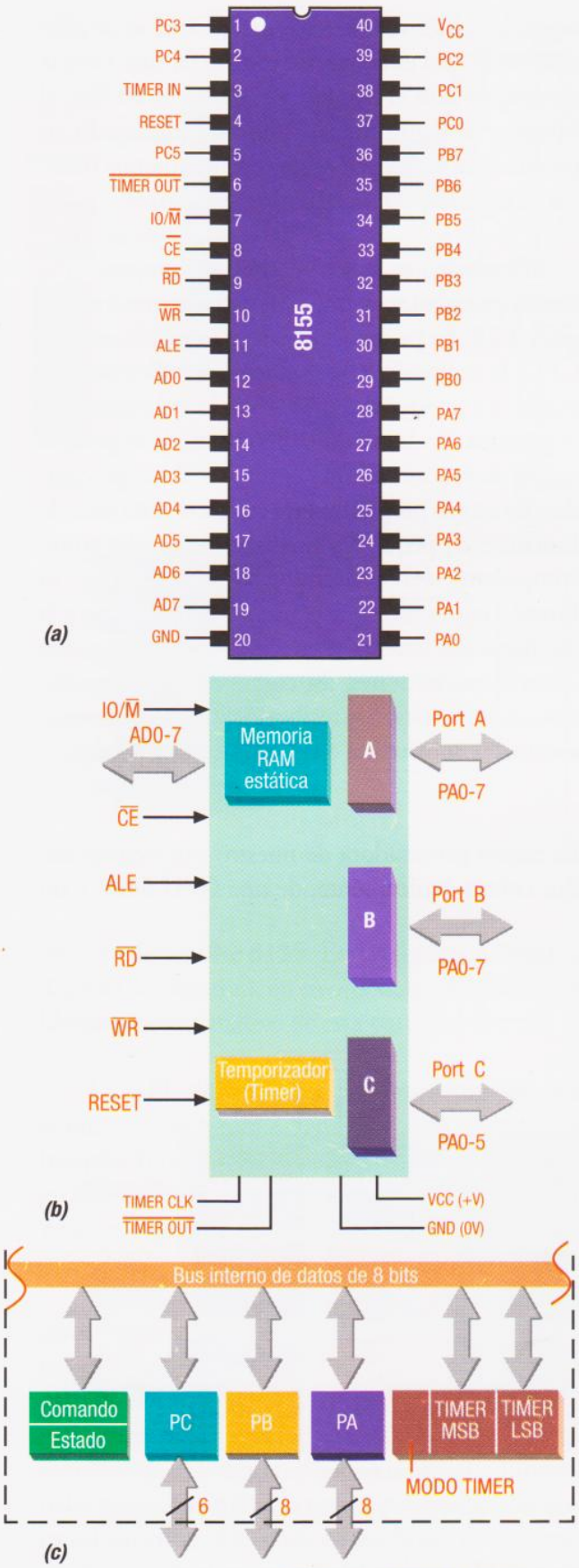


Figura 2.22 *Circuito de soporte 8155*

cesador, utilizando el circuito integrado **8155**. En la **figura 2.22** se muestra la configuración de pines, el diagrama interno de bloques y la estructura de registros de este circuito.

El **8155** puede demultiplexar internamente el bus de datos y direcciones del **8085**. Por esta razón, sólo se necesitan ocho líneas, desde AD0 hasta AD7, para transmitir datos y direcciones. Las líneas IO/$\overline{M}$D, $\overline{RD}$ y $\overline{WR}$ se utilizan para el control de la memoria y las líneas ALE y RESET para el control interno.

Para efectos de la interfaz se utilizan 22 líneas programables como entradas o salidas y organizadas en tres grupos así: desde PA0 hasta PA7 para el puerto A, desde PB0 hasta PB7 para el puerto B y desde PC0 hasta PC5 para el puerto C. Las líneas TI (pin 3) y $\overline{TO}$(pin 6) son, respectivamente, la entrada y la salida del temporizador.

La memoria RAM del **8155** tiene asignadas las direcciones 0000H hasta 00FFH y es accesible externamente mediante los ocho *bits* de direccionamiento de más bajo orden cuando las líneas $\overline{CE}$ e IO/$\overline{M}$ son **bajas** y cualquiera de las líneas $\overline{RD}$ ó $\overline{WR}$ es **baja**. Los ocho *bits* de direccionamiento de más bajo orden se almacenan en un registro interno durante los flancos de bajada de la señal ALE.

La capacidad de memoria RAM del **8155** (256 *bytes*) es, generalmente, más que suficiente para sistemas pequeños. En la mayoría de los casos esta memoria se utiliza para almacenamiento temporal de datos y resultados, así como de información de registros y direcciones.

Acceso a los puertos

Los puertos A, B y C se pueden utilizar como tres puertos de entrada/salida independientes; estos puertos son accesibles externamente a través de las 8 líneas de más bajo orden del bus de direcciones cuando $\overline{CE}$ es **baja**, $\overline{RD}$ o $\overline{WR}$ es **baja** e IO/$\overline{M}$ es **alta**. La **figura 2.23** muestra las

direcciones internas de cada registro y de cada puerto en el **8155**. Por ejemplo, para seleccionar el puerto B, A1 debe ser **alta** (**1**) y A2 y A0 deben ser **bajas** (**0**).

Diseño de un sistema mínimo con 8085 y 8155

Podemos decir que un ***sistema mínimo*** con microprocesador es un dispositivo constituido por una serie mínima de elementos que permiten el desarrollo de algún tipo de tarea o aplicación en el mundo real de manera autónoma, inteligente y siguiendo las pautas dadas por un programa. En el estudio de los microcontroladores, es muy importante entender qué es un sistema mínimo con microprocesadores, ya que es precisamente esta configuración básica la que se ha reemplazado con esta nueva tecnología.

Como ejemplo mostraremos el diseño de un sistema microcomputador mínimo desarrollado por CEKIT alrededor de una CPU **8085** y un circuito de soporte **8155**. Aunque el 8085 es un microprocesador de ocho *bits* de las primeras generaciones, lo hemos utilizado en este ejemplo por la facilidad para entenderlo con fines didácticos según los objetivos de este curso. De él se originó el 8086 el cual es la base de toda la familia Intel la que siguió con el 80186, el 80286, el 80386, el 80486 y el Pentium. Si usted está interesado en profundizar en este tema, le recomendamos nuestro **Curso Básico de Microprocesadores**.

El sistema se ha dividido en dos tarjetas: una tarjeta procesadora y una tarjeta programadora, **figura 2.23**. La primera contiene, esencialmente, la CPU, la memoria RAM, el temporizador programable, los puertos de entrada/salida y el registro de comandos. En la memoria RAM reside el programa de instrucciones que ejecuta el microprocesador. La tarjeta programadora consta de una serie de funciones de pequeña y mediana escala tales como compuertas, *drivers*, decodificadores y registros. Su función es permitir que el usuario tenga acceso a la memoria del sistema y pueda cargar en la misma datos e instrucciones, así como visualizar resultados. La información se recibe desde un teclado hexadecimal y se visualiza en diodos LED y *displays*.

Diseño de la tarjeta procesadora

La tarjeta procesadora de nuestro microcomputador consta, básicamente, de una CPU **8085** y un

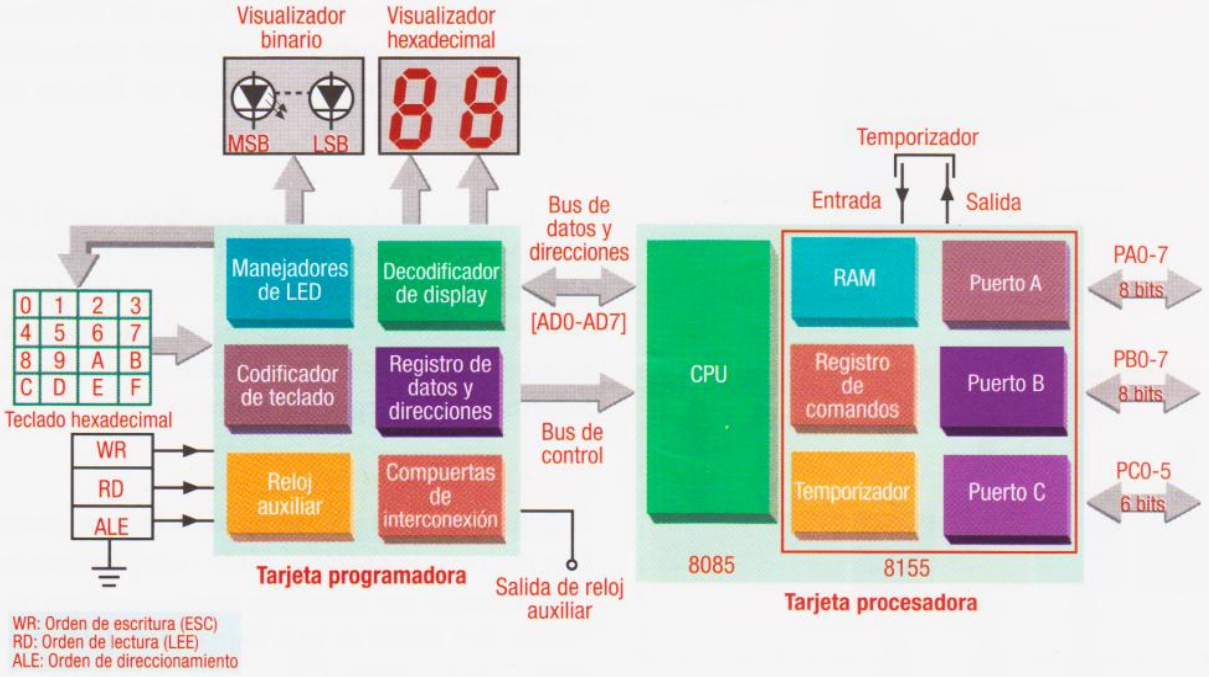


Figura 2.23 *Sistema mínimo*

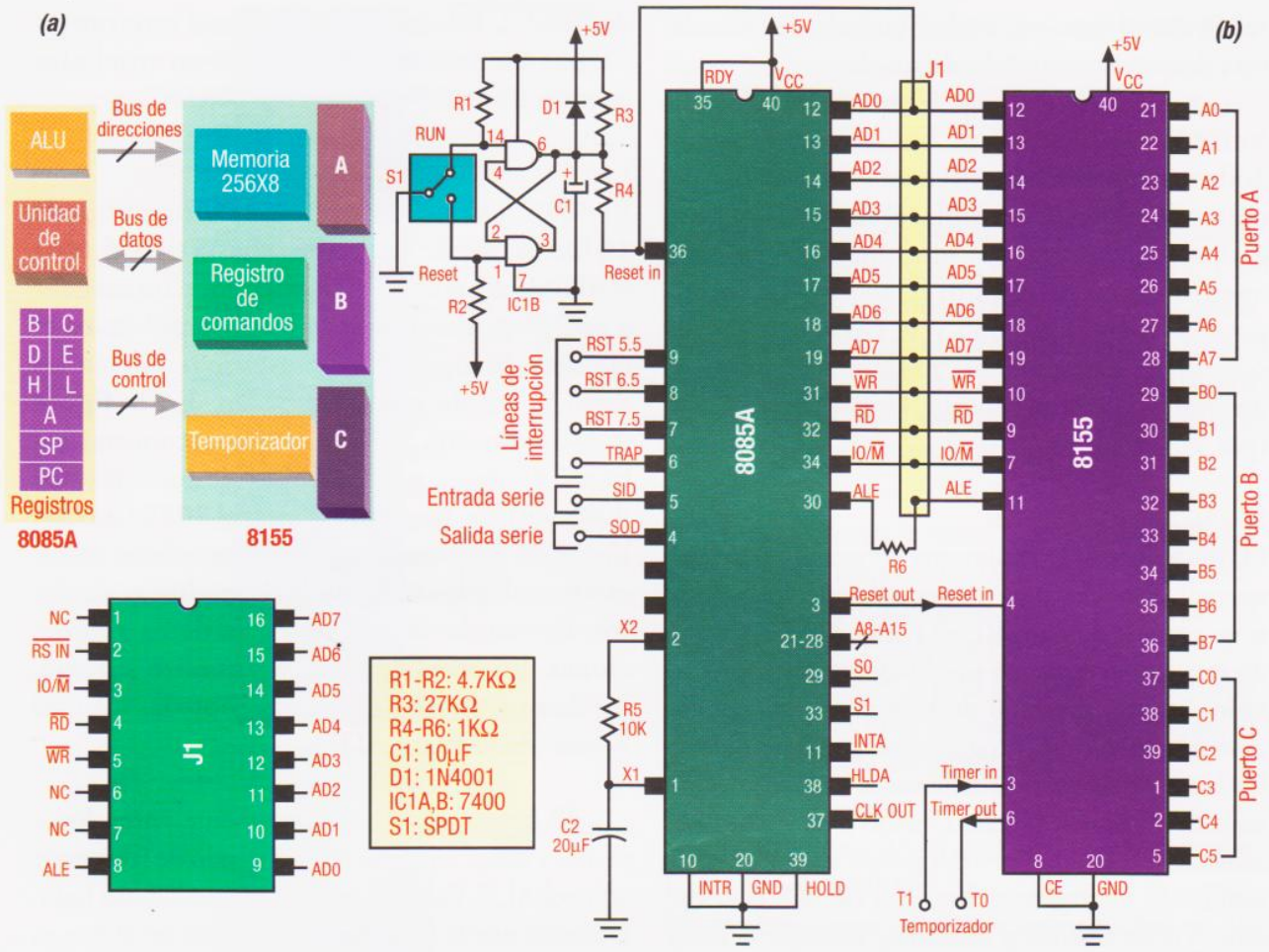


Figura 2.24 (a) Diagrama de bloques (b) Diagrama esquemático

circuito de soporte **8155**. En las **figuras 2.24(a)** y **2.24(b)** se muestran, en su orden, los diagramas de bloques y esquemático de esta parte del sistema.

La CPU 8085, cuya configuración interna se ilustró en la figura 2.5, proporciona, entre otros bloques funcionales, una unidad aritmético-lógica (ALU) de ocho *bits*, una unidad de control y varios registros accesibles al usuario.

Aunque el **8085**, por disponer de un bus de direcciones de 16 *bits*, puede direccionar un espacio de memoria de 64K (65536 *bytes*), los 256 *bytes* de RAM disponibles en nuestro sistema son suficientes para soportar la mayoría de programas sencillos de aplicación. Todos los bloques funcionales que constituyen la tarjeta procesadora se comunican entre sí a través de los buses de control, de datos y de direcciones del sistema. El bus de control, como su nombre lo indica, transporta las señales de control (ALE, $\overline{WR}$, $\overline{RD}$, IO/$\overline{M}$, etc) que necesita la CPU para comunicarse o *dialogar* con la memoria y los puertos I/O.

A través del bus de direcciones, el **8085** selecciona de manera inequívoca la posición de memoria o el puerto I/O desde el cual trae o hacia el cual dirige la información del bus de datos. Puesto que el sistema tiene solamente 256 *bytes* de memoria, sólo se requieren ocho líneas de direccionamiento (las menos significativas); las ocho restantes (las más significativas), no se utilizan. En el **8085** las ocho líneas de direccionamiento de más bajo orden (A0 hasta A7), se multiplexan internamente con las ocho líneas del bus de datos (D0 hasta D7) formando un bus único de datos y direcciones. El **8155** decodifica o demultiplexa la información transportada

sobre estas líneas para saber cuándo se trata de una dirección y cuándo de un dato.

El circuito de reloj

Todo microprocesador necesita un generador de pulsos de reloj para controlar el contador de programa y sincronizar sus funciones internas. En nuestro caso, este circuito de reloj viene incorporado en la CPU **8085** y su frecuencia puede controlarse externamente mediante un cristal o una red RC externa conectada entre las líneas X1 (pin 1) y X2 (pin 2).

El circuito de *reset*

El **8085** puede ser fácilmente conectado o desconectado por *hardware* manejando adecuadamente su entrada de *reset* ($\overline{\text{RESET IN}}$), pin 36. Cuando esta línea recibe un nivel bajo, el microprocesador ingresa al estado de *reset*. Bajo esta condición, el contador de programa se carga con ceros y los buses de datos y direcciones se sitúan en el estado de alta impedancia. Debido a su naturaleza asincrónica, la señal de *reset* puede afectar el contenido de los registros internos de la CPU, incluido el de estados o banderas. Cuando la línea $\overline{\text{RESET IN}}$ se hace **alta**, el **8085** procede a ejecutar el programa almacenado en la memoria.

En el momento de aplicar energía al sistema, el condensador C1 está descargado e inyecta, a través de R4, un nivel **bajo** a la entrada $\overline{\text{RESET IN}}$ del **8085**. Como resultado, este último pasa al estado de *reset* descrito anteriormente. El condensador comienza entonces a cargarse a través de R3. Cuando el voltaje entre los terminales de C1 alcanza un valor cercano a +5V, la entrada $\overline{\text{RESET IN}}$ recibe un nivel alto y el 8085 sale del estado de *reset*, comenzando la ejecución de las instrucciones a partir de la posición 0000H. El diodo D1 descarga rápidamente el condensador cuando se desconecta momentáneamente la fuente de alimentación, incluso como resultado de una falla en la alimentación. El circuito constituido por S1, R1, R2, IC1A e IC1B forma un eliminador de rebote cuyo objetivo es aplicar un nivel **bajo** a la entrada $\overline{\text{RESET IN}}$ del **8085** cuando el interruptor S1 está en posición "RESET" y un nivel **alto** cuando S1 está en la posición "RUN".

Los buses de datos y direcciones

El **8085**, por poseer un bus de direcciones de 16 líneas, puede direccionar hasta 65536 *bytes* (64K) de memoria. Sin embargo, para acceder a las 256 posiciones de RAM disponibles en el circuito de soporte **8155** sólo se requieren las ocho líneas de más bajo orden. En la **figura 2.25** se muestra la forma de interconectar los buses de datos y direcciones de los circuitos 8085 y 8155. Recuerde que en el **8085** las ocho líneas de direccionamiento de más bajo orden están multiplexadas con las ocho líneas de datos, formando un bus único de datos y direcciones. En otras palabras, en nuestro sistema, los datos y direcciones viajan sobre las mismas líneas (AD0 hasta AD7).

El **8155** distingue internamente entre direcciones y datos interpretando el estado de la línea de control ALE. Cuando esta última es de nivel **bajo**, entiende que la información presente en el bus es un dato y cuando es de nivel **alto** es una dirección.

Cuando el bus AD actúa como bus de datos, las líneas AD0 a AD7 son bidireccionales, es decir, pueden transportar información desde o hacia el microprocesador. Cuando este bus se convierte en bus de direcciones, las líneas AD0 hasta AD7 son unidireccionales y llevan información de la CPU hacia la memoria o hacia los puertos de entrada/salida.

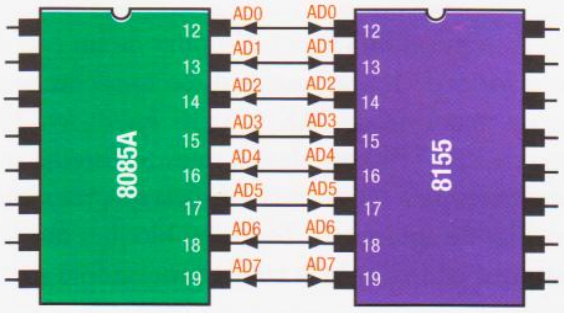


Figura 2.25 *Conexión multiplexada del bus de datos y direcciones*

El bus de control

Como se muestra en la **figura 2.26**, la CPU **8085** se comunica con el circuito de soporte **8155** a través de cinco líneas de control: RESET, ALE, $\overline{WR}$, $\overline{RD}$ e IO/$\overline{M}$. Otras líneas de control como HOLD, HLDA, READY, etc., no son necesarias para la operación de nuestro sistema mínimo pero sí se requieren en sistemas más grandes para ejecutar otras funciones.

La salida de *reset* del **8085** (RESET OUT, pin 3), activa en **alto**, le comunica a la entrada de *reset* del **8155** (RESET IN, pin 4) que el microprocesador está en condición de *reset*, es decir que no está ejecutando el programa de aplicación. Cuando esto sucede, el **8155** entra también en estado de *reset* y define todas las líneas de sus puertos como entradas.

La línea ALE (*Address Latch Enable*: habilitador del seguro de direcciones) del **8085** (pin 10) está conectada a la línea ALE del **8155** (pin 11). Cuando esta señal es **alta**, la CPU le informa a la memoria interna del **8155** que la información binaria disponible en el bus AD es una dirección.

Como resultado de la señal ALE, el **8155** almacena en un registro interno, durante el flanco de bajada, la palabra de dirección. En el siguiente ciclo de máquina, el microprocesador puede proceder entonces, a leer o a escribir en la posición de memoria direccionada. En este caso, la información se transmite sobre el bus AD, convertido ahora en el bus de datos del sistema.

Figura 2.26 *Conexión del bus de control*

La línea $\overline{WR}$ (*write*: escritura) del **8085** (pin 31) está conectada a la línea $\overline{WR}$ (pin 10) del **8155**. Cuando esta línea se hace **baja**, la CPU le indica a la memoria que el dato o la instrucción presente en el bus AD va a ser escrita en la posición de memoria previamente seleccionada. La línea $\overline{RD}$ (*read*: lectura) del **8085** (pin 32) está conectada a la línea $\overline{RD}$ (pin 9) del **8155**. Cuando esta línea se hace **baja**, la CPU le indica a la memoria que va a proceder a leer el contenido de la dirección previamente seleccionada y a situar la información en el bus AD.

Finalmente, la línea IO/$\overline{M}$ (*Input/output-memory*: entrada/salida-memoria) del **8085** (pin 34) está conectada a la línea IO/$\overline{M}$ (pin 7) del **8155**. Cuando esta línea se hace **alta**, el **8155** reconoce que la palabra de dirección presente en el bus de datos corresponde a un registro interno o a un puerto I/O. Cuando la línea IO/$\overline{M}$ se hace **baja**, la CPU **8085** le informa al circuito de soporte **8155** que la dirección presente en el bus AD corresponde a una de las 256 posiciones de memoria a las que es posible tener acceso. Estas localizaciones tienen asignadas direcciones entre 00H y FFH.

Diseño de la tarjeta programadora

Desde un punto de vista conceptual, el circuito de la figura 2.24 es una microcomputadora completa porque posee una unidad central de proceso, una unidad de memoria y puertos de entrada/salida. Sin embargo, le falta lo más importante: un programa. Sin un programa que le indique lo que debe hacer, un microcomputador es sólo una masa inanimada de componentes electrónicos.

Para almacenar en la memoria los datos e instrucciones que constituyen el programa de nuestra microcomputadora, necesitamos asumir temporalmente el control del circuito de soporte **8155** y aislar el microprocesador **8085** del resto del sistema. De este modo, podemos introducir información en la memoria sin intervención de la CPU. Esta es, precisamente la función de la

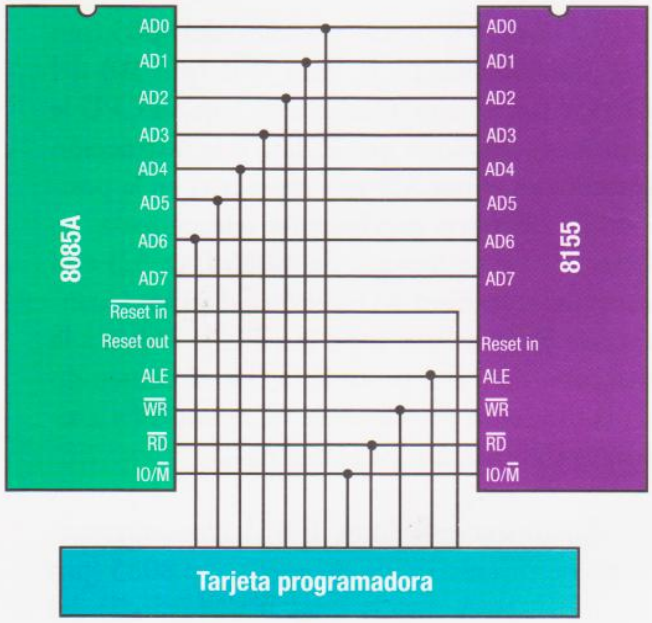


Figura 2.27 *Conexión de la tarjeta programadora*

tarjeta programadora. En la **figura** 2.27 se ilustra la forma de conectar esta tarjeta a la tarjeta procesadora. Cuando el interruptor S1 de esta última se sitúa en posición "RESET", las líneas de datos y direcciones AD0 hasta AD7 del **8085** adoptan el estado de alta impedancia y quedan flotantes. Lo mismo sucede con las líneas de control ALE, $\overline{WR}$, $\overline{RD}$ e IO/$\overline{M}$. La línea RESET IN del **8085** es de nivel **bajo**.

Bajo estas condiciones, desde la tarjeta programadora podemos acceder libremente a la RAM del **8155** y escribir en la misma el programa que debe ejecutar el microprocesador. Así mismo, podemos direccionar cualquier posición de memoria y leer su contenido. Una vez cargado el programa, el control del **8155** debe retornar al **8085** para que este último lo ejecute (S1 = "RUN"). En la **figura 2.28** observamos el aspecto físico del sistema mínimo desarrollado alrededor del microprocesador **8085** y del circuito integrado **8155**.

Notas finales

En esta lección se ha hecho un repaso general de los aspectos más importantes relativos al mundo de los microprocesadores. Hemos analizado su arquitectura interna, su funcionamiento y la forma de interconectarlos dentro de un sistema para que puedan ejecutar tareas que involucren el manejo de variables reales y cierto grado de inteligencia. Hemos visto que estos dispositivos requieren de componentes adicionales para ejecutar una tarea específica y que poseen las facilidades para direccionar o manejar gran cantidad de periféricos como memorias, puertos y muchos otros dispositvos. Debido a esta capacidad, los microprocesadores son el elemento fundamental dentro de los equipos de computación modernos, donde se manejan enormes cantidades de memoria y gran variedad de periféricos.

Los microcontroladores son dispositivos electrónicos que agrupan en un solo circuito integrado todas estas características, aunque con cierta limitaciones de memoria y periféricos. Sin embargo, su funcionamiento interno sigue siendo básicamente el mismo. Por esta razón hemos querido entregar esta corta introducción a los microprocesadores, para dar una idea de todo el desarrollo tecnológico que involucra esta tecnología y sentar las bases para el estudio de estos elementos.

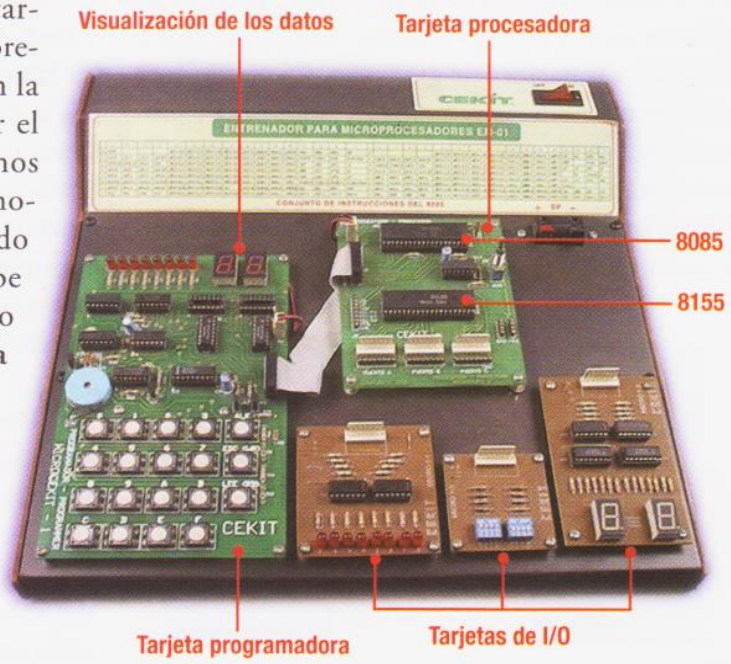
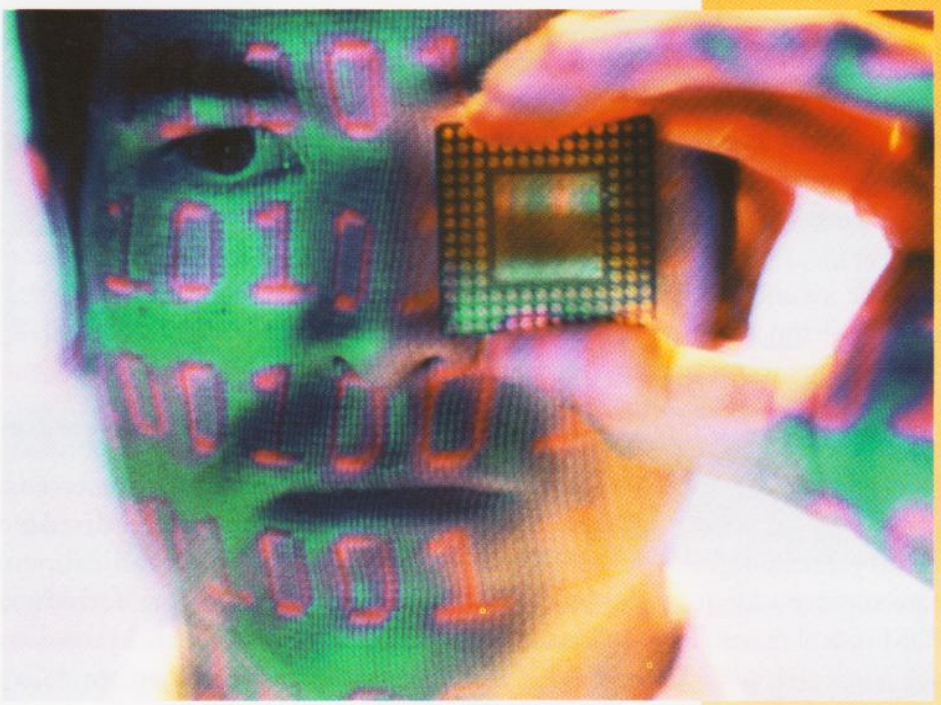


Figura 2.28 *Sistema microcomputador mínimo con el* ***8085***

LECCIÓN 3



LOS MICROCONTROLADORES

En los últimos años, un componente electrónico ha revolucionado las técnicas de diseño; se trata del microcontrolador. Es un circuito integrado programable que contiene todos los elementos necesarios para controlar un sistema.

A partir de esta lección, iniciamos un estudio detallado de cada una de las partes y módulos internos que componen un microcontrolador, desde los más básicos hasta los más especializados, al igual que algunas técnicas y algoritmos que permiten su manejo e inclusión dentro de un proyecto.

Al analizar los sistemas basados en microprocesadores mostrados en las lecciones anteriores, se puede deducir que la estructura de un sistema de microcomputadora para aplicaciones de medición o control, tiene un carácter constante, es decir, siempre existirá esa misma estructura (CPU, memorias, puertos, buses, etc.). No es difícil comprender, entonces, porqué los fabricantes de circuitos integrados decidieron producir un supercomponente que contenga todos los elementos de una computadora en un solo circuito integrado.

A mediados de los años 70, los fabricantes de circuitos integrados dieron a conocer un nuevo dispositivo denominado **microcontrolador** el cual contenía toda la estructura de una microcomputadora, es decir, unidad de proceso (CPU), memoria RAM, memoria ROM y circuitos de entrada/salida. El primer dispositivo de esta naturaleza fue lanzado por Texas Instruments y se denominó TMS 1000, el cual fue el primer microprocesador que incluyó suficiente RAM y ROM para el desarrollo de aplicaciones.

El microcontrolador se concibió como un dispositivo programable que puede ejecutar un sinnúmero de tareas y procesos. Desde ese momento, el diseño de productos electrónicos cambió radicalmente. Circuitos lógicos, manejo de periféricos, temporizadores y estructura de computadoras, todo programable y alojado en un solo circuito integrado; es decir, una pequeña computadora para todas las aplicaciones.

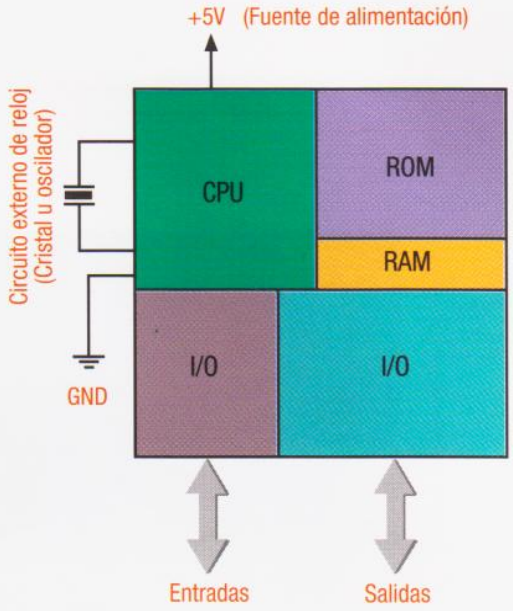


Figura 3.1 *Idea general de un microcontrolador*

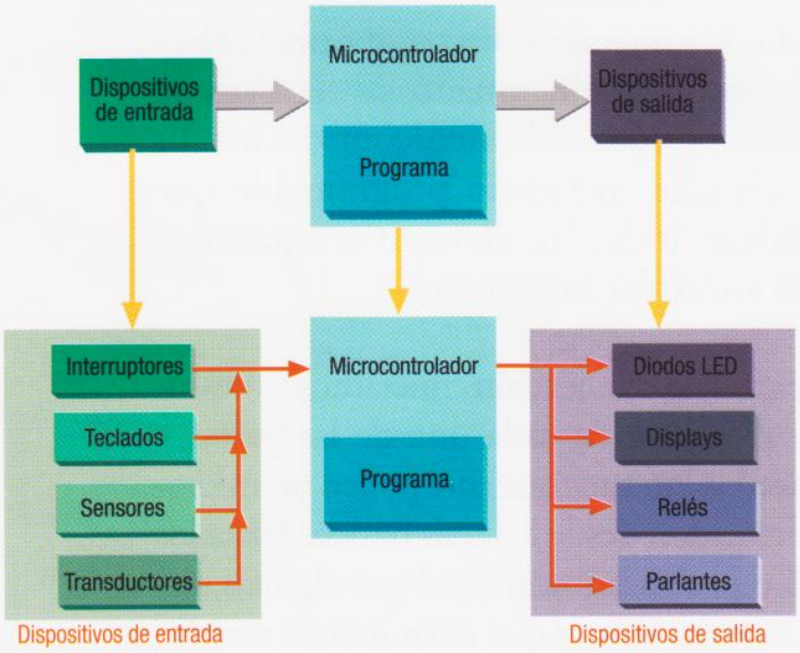


Figura 3.2 *Diagrama general de un sistema basado en un microcontrolador*

Ahora, para un diseñador, la idea de una microcomputadora, en aplicaciones de control o en la fabricación masiva de aparatos modernos, se asimila al esquema mostrado en la **figura 3.1**, donde tenemos una fuente de alimentación, un circuito de reloj y un microcontrolador. Una vez configurado el sistema en cuanto al *hardware*, solamente se requiere grabar el programa en la memoria ROM; los puertos ya están listos para conectarse al mundo exterior. Un microcontrolador puede definirse entonces como un sistema completo de cómputo, que incluye una CPU, memoria RAM y ROM, un oscilador y puertos I/O, todo dentro de un mismo circuito integrado.

Sistemas microcontrolados

Aunque un usuario común no lo perciba, hoy en día es muy común encontrar uno o varios mi-

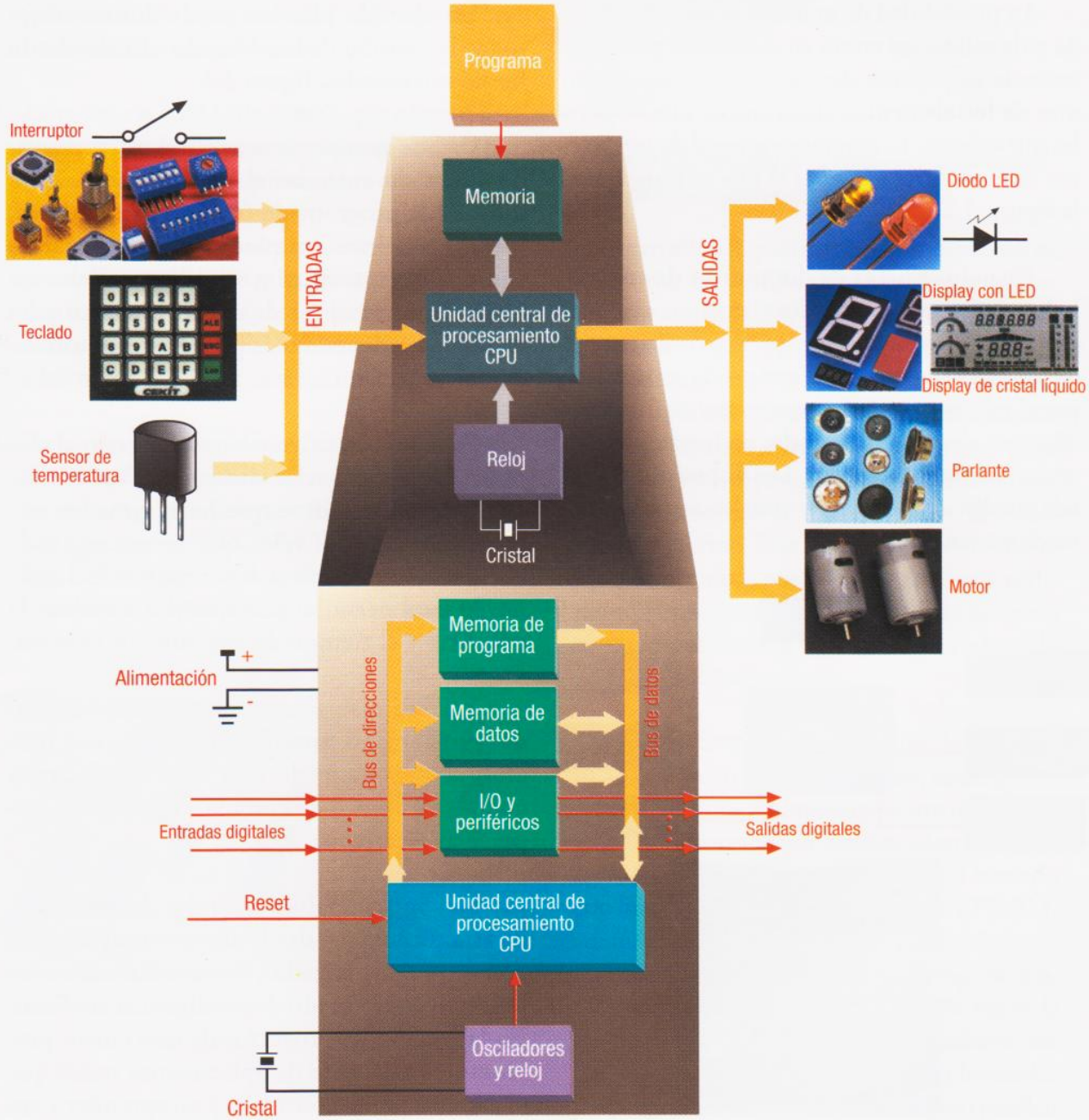


Figura 3.3 *Estructura básica de un microcontrolador y de los dispositivos que pueden hacer parte de un sistema*

crocontroladores en todo tipo de aplicaciones desde los electrodomésticos y aparatos caseros como las cafeteras, los hornos microondas, las videograbadoras y las alarmas, hasta los circuitos de control sofisticados de los automóviles, los aviones, las naves espaciales, los barcos, los sistemas de automatización industrial, etc. Las aplicaciones para los microcontroladores son infinitas; el único límite es la imaginación.

La posibilidad de manejar señales de entrada y de salida, así como su capacidad para procesar datos y tomar decisiones, lo convierte en uno de los elementos electrónicos más versátiles que existen. El diagrama general de un sistema microcontrolado sería el que se muestra en la **figura 3.2.**

Cuando se habla de dispositivos de entrada, se hace referencia a todos los elementos que pueden cambiar su estado ante una determinada condición y generar una señal que pueda ser utilizada por el microcontrolador para tomar alguna decisión; por ejemplo, un teclado, un interruptor, un sensor, un transductor, etc. Los dispositivos de salida pueden ser indicadores visuales, audibles o actuadores, entre otros.

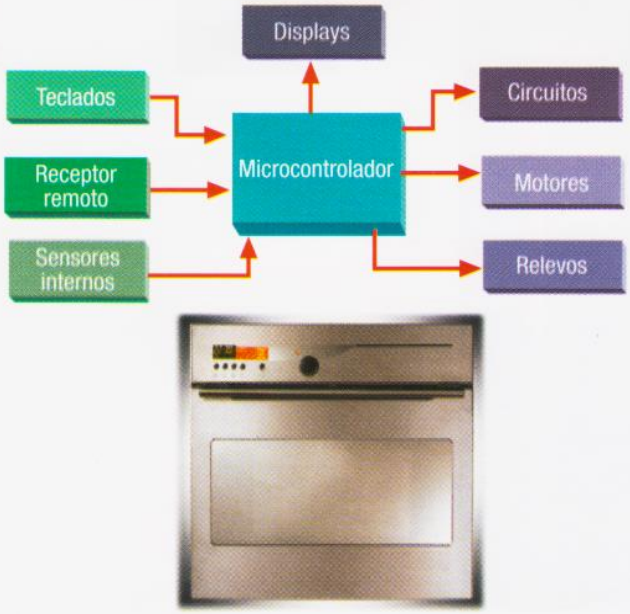


Figura 3.4 *Ejemplo del uso de un microcontrolador en un horno microondas*

Los indicadores visuales más comunes son las pantallas de cristal líquido, los diodos LED, las lámparas incandescentes, etc. Los indicadores audibles pueden ser parlantes, zumbadores, etc.

Los actuadores son muy importantes ya que se utilizan para manejar las "cargas" como los relés, las electroválvulas, los motores, etc., **figura 3.3.**

Un ejemplo práctico puede ilustrar mejor todas las posibilidades. Veamos el caso de un horno microondas, **figura 3.4.**

En este ejemplo tenemos tres clases de dispositivos de entrada: el teclado, por medio del cual se programa la función que se desea, los sensores internos para controlar variables como la temperatura y, si el horno tiene esa función, un receptor de control remoto para captar las órdenes dadas por el usuario sin accionar el teclado.

Como dispositivos de salida están: el *display*, que puede ser de diodos LED o de cristal líquido, el motor que hace girar los alimentos y algunos relés que se encargan de controlar o desactivar los circuitos del horno, lo cual permite, por ejemplo, cambiar la potencia y el tiempo de cocción, entre otros.

En este caso tenemos un microcontrolador programado específicamente para esta función acompañado de una serie de circuitos complementarios de entrada y salida y sus respectivas interfaces.

Hay innumerables ejemplos de aplicación de esta tecnología, desde algunos muy simples como un temporizador, hasta sofisticados robots con cierto grado de inteligencia artificial. En la sección de proyectos de este curso, presentamos una serie de aplicaciones reales que el alumno podrá construir, y así aprender y verificar el poder y la versatilidad que ofrecen estos dispositivos.

Diferencias entre sistemas basados en microprocesador y en microcontrolador

Existen varias diferencias importantes entre estas dos tecnologías, cada una tiene ventajas y desventajas, y su uso depende de las necesidades de cada aplicación. Para aclarar las ideas sobre este tema, veamos cuales son estas diferencias:

1. La CPU del microcontrolador es más simple y sus instrucciones están orientadas, principalmente, a la operación de cada una de las líneas de entrada y salida.

2. La memoria RAM (de datos), que ofrecen los microcontroladores, es de baja capacidad. La razón es simple: las aplicaciones de control e instrumentación comunes no necesitan almacenar grandes cantidades de información temporal. En cambio, los microprocesadores pueden acceder, a través de los buses, a grandes bancos de memoria RAM externa de acuerdo a las necesidades del sistema.

3. En los microcontroladores, la memoria ROM (de programa) es limitada. Por lo general, no mayor a 4 Kilobytes. De esta memoria hay tres tipos: EPROM, OTP y EEPROM, **figura 3.5**. Los microcontroladores con memoria EPROM se utilizan básicamente para el desarrollo de prototipos; con este tipo de memoria, es posible programar y reutilizar el microcontrolador a la manera de las memorias 2716, 2732, etc., esto es, borrando el programa mediante un haz de luz ultravioleta aplicado sobre una ventanita. En cambio, los circuitos integrados OTP (*One Time Programable*) o programables una sola vez, no permiten que se altere o borre la información escrita. Este tipo de dispositivos es más económico, por lo tanto es el más utilizado para la producción en serie de aparatos electrónicos, cuando el prototipo ya ha sido completamente probado.

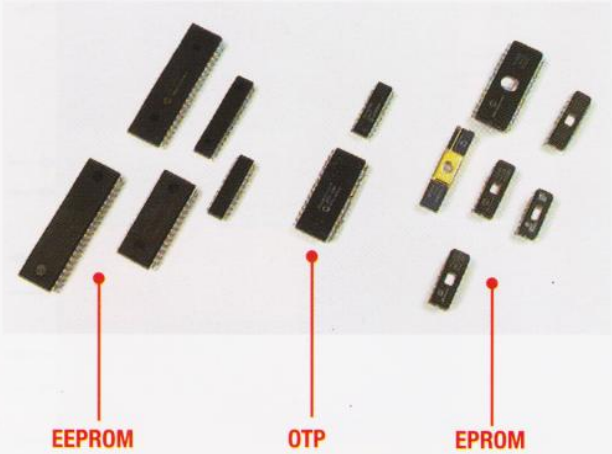


Figura 3.5 *Tipos de memoria en los microcontroladores*

Figura 3.6 *Dispositivo de luz ultravioleta para el borrado de microcontroladores con memoria EPROM*

Los dispositivos EEPROM, que han hecho su aparición en los últimos años, se caracterizan por permitir el almacenamiento de un programa y su borrado por medios eléctricos. Este método es mucho más rápido que el borrado con luz ultravioleta de las memorias EPROM, además de no necesitar dispositivos adicionales como el que se muestra en la **figura 3.6.** Una variedad de la memoria EEPROM es la memoria FLASH; los microcontroladores con este tipo de memoria, también son borrables y reprogramables eléctricamente, pero pueden soportar un número mucho mayor de estos procedimientos. En un sistema con micropro-

cesador, se pueden tener memorias ROM externas de diferentes tecnologías y capacidades, de la misma forma que las memorias RAM.

4. Con los microcontroladores no es necesario diseñar complejos circuitos decodificadores porque el mapa de memoria y de puertos I/O están incluidos internamente. Por la razón anterior, el circuito impreso de las aplicaciones es muy simple y, en algunas ocasiones, puede ser de una sola capa lo cual es una ventaja sobre los microprocesadores.

5. La mayoría de los microcontroladores no tienen accesible al usuario los buses de direcciones, de datos y de control de la CPU. Algunos modelos lo hacen a través de los puertos de entrada/salida (I/O), utilizando señales especiales de sincronización. Estos buses y señales se pueden emplear para construir expansiones de memoria RAM y ROM por fuera del microcontrolador. Como ya vimos, esta función es mucho más fácil de utilizar en los microprocesadores.

6. La velocidad de operación de los microcontroladores es más lenta que la que se puede lograr con los sistemas de microprocesadores. Sin embargo, actualmente existen microcontroladores que funcionan por encima de los 50 MHz lo que es suficiente para muchas aplicaciones.

7. De manera similar a los sistemas utilizados con los microprocesadores para escribir, ensamblar y depurar programas en lenguaje de máquina, se requiere un sistema de desarrollo para cada familia de microcontroladores. Está compuesto por un paquete de *software* con editor, ensamblador y simulador de programas y, al mismo tiempo, se necesita un *hardware* para "quemar" o almacenar el programa en la memoria del microcontrolador.

En resumen, algunas de las principales ventajas cuando hacemos el diseño de un circuito utilizando un microcontrolador son:

- El circuito impreso es más pequeño ya que muchos de los componentes se encuentran dentro del circuito integrado.
- El costo del sistema es mucho menor, al reducir el número de componentes.
- Los problemas de ruido que pueden afectar los sistemas con microprocesador se eliminan, debido a que todo el sistema principal se encuentra en un solo encapsulado.
- El tiempo de desarrollo de un sistema se reduce notablemente.

Cuando una aplicación o problema que se va solucionar, sobrepase las características de un microcontrolador como capacidad de memoria, velocidad de proceso, número de pines o líneas de entrada y salida, etc., se debe recurrir a un sistema con microprocesador o a una microcomputadora completa.

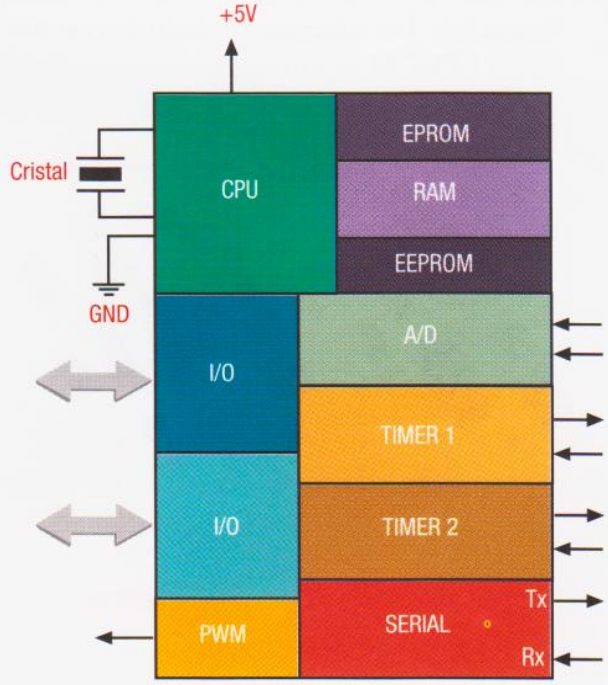


Figura 3.7 *Módulos especializados en los microcontroladores*

Un microcontrolador para cada gusto

Muchas aplicaciones requieren algo más que las simples entradas y salidas de tipo digital. Pensando en ello, los fabricantes de microcontroladores han adicionado internamente algunos circuitos especiales en algunos de sus modelos. Las posibilidades son amplísimas y el usuario puede escoger a la carta. A continuación, enunciaremos algunas de las posibilidades, las cuales iremos estudiando progresivamente en este curso.

- Si los fenómenos que se necesitan medir o controlar son de naturaleza análoga como temperatura, voltaje, etc., se debe disponer de un convertidor análogo-digital (A/D) de varios canales.

- Si hay que medir períodos de tiempo o generar, en las salidas del circuito, tiempos, tonos o frecuencias, hay que tener uno o más contadores programables (*timers*).

- Si la información obtenida en un proceso de medida o control, o el resultado de los cálculos del programa, se deben enviar a otro microcontrolador, a una computadora o a una red, es muy útil tener un circuito RS-232 para comunicaciones seriales.

- Para quienes requieren salidas PWM (modulación por ancho de pulso) para el control de motores CC o cargas resistivas, existen microcontroladores que pueden ofrecer varias de ellas.

- Cuando se requiere atender eventos en tiempo real o existen procesos que no dan espera, se debe utilizar la técnica llamada de "interrupciones". Cuando una señal externa activa una línea de interrupción, el microcontrolador deja de lado la tarea que se encuentra ejecutando para atender una situación especial y luego puede regresar a continuar con la labor que estaba desempeñando.

Referencia	Memoria ROM	Memoria RAM	Timers
8051	4K	128	2
8052	8K	256	3
8031	Externa	128	2
8032	Externa	256	3
8751	4K (Eprom)	128	2

Tabla 3.1 *Microcontroladores de la familia INTEL 8051*

En la **figura 3.7** se presenta el diagrama de un posible circuito con microcontrolador que incluye la mayoría de los elementos que se utilizan en aplicaciones de medición, instrumentación y control.

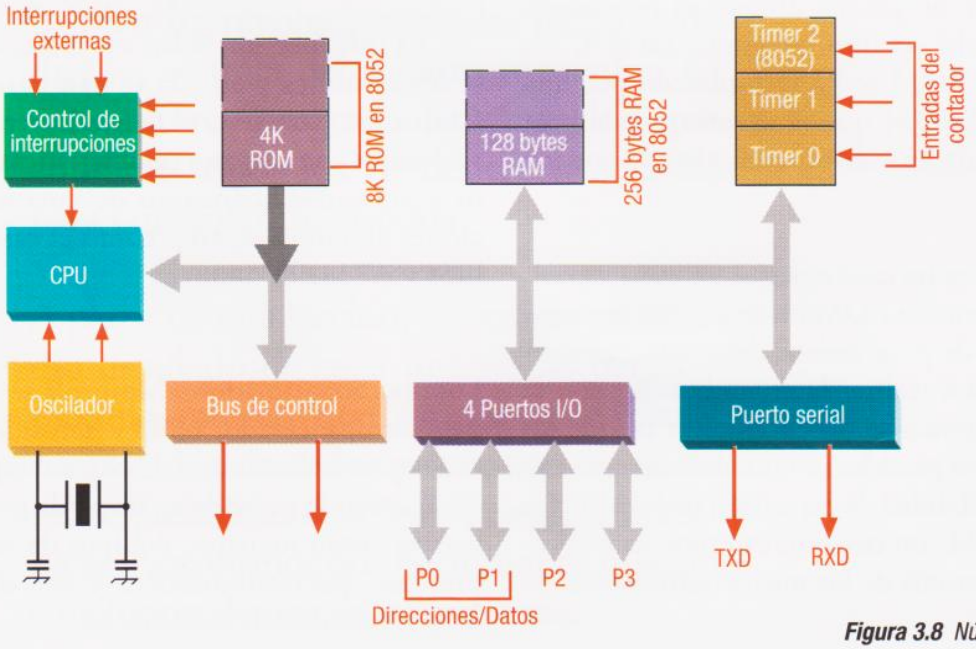


Figura 3.8 *Núcleo del 8051*

Principales familias de microcontroladores

Actualmente existen en el mercado varias marcas reconocidas como las más importantes dadas sus características, comercialización, soporte técnico, difusión, usos en la industria y en la fabricación de productos de consumo masivo, etc. Entre ellas están Motorola, Microchip, Intel, Philips, National y ATMEL.

La familia Intel 8051

El primer microcontrolador de Intel fue el 8048, en el cual la memoria RAM estaba dentro del circuito integrado, pero el programa debía almacenarse externamente; posterior a él, en los años 80, nació el 8051. Esta familia de microcontroladores ha sido una de las más difundidas en el mundo e incluye varios miembros, cada uno de ellos acondicionado para aplicaciones específicas. Todas las versiones existentes están conformadas según un núcleo básico (*8051 Core*) cuyo diagrama de bloques se muestra en a **figura 3.8.**

Tienen la misma CPU, memoria RAM, temporizadores, puertos paralelos y entradas/salidas de tipo serial, según la distribución que se muestra en la **tabla 3.1.**

El 8051 tiene 4 kilobytes de memoria ROM que se deben programar durante el proceso de fabricación del circuito integrado. En el 8751, la memoria ROM se ha reemplazado por una memoria EPROM que el usuario puede programar con un aparato especial y borrar con luz ultravioleta.

El 8031 es un caso especial; no tiene capacidad para memoria ROM interna y, por lo tanto, la memoria de programa se debe colocar externamente. Para realizar la comunicación con la memoria externa, se deben emplear tres de los cuatro puertos paralelos de entrada/salida que tiene. Esta posibilidad de expansión de la memoria RAM y ROM con circuitos externos, es una característica propia de los microcontroladores de esta familia.

Mapa de memoria. Los componentes de esta familia de microcontroladores tienen una característica en común que los hace diferentes al comportamiento convencional de un microprocesador: tienen un espacio para las direcciones de datos (lectura/escritura) y otro para las direcciones de programa o de lectura solamente.

La memoria de programa puede ser hasta de 64 *Kbytes*. En las versiones que tienen ROM y EPROM, los 4K, 8K o 16K inferiores de programa, están alojados en el microcontrolador.

Conjunto de instrucciones. Todos los miembros de la familia ejecutan las mismas instrucciones. Este conjunto de instrucciones ha sido diseñado pensando en optimizar aplicaciones de control de 8 bits. Ofrece una variedad de modos de direcionamiento para accesar la memoria RAM interna con el fin de facilitar las operaciones de 8 bits en una estructura de datos pequeña.

El conjunto de instrucciones proporciona un soporte muy amplio para variables de un *bit*. De esta manera, se puede lograr la manipulación en los sistemas lógicos y de control, que necesitan operaciones de álgebra de Boole.

La información sobre el núcleo del 8051 se hizo pública y en ella se han basado otros fabricantes para llevar al mercado versiones actualizadas con mejores características en cuanto a velocidad, consumo de corriente, funciones adicionales, etc., como el caso de Phillips y ATMEL.

Motorola

Los microcontroladores Motorola se derivaron del microprocesador 6800 y han sido optimizados para aplicaciones de control especializado, formando parte de aparatos de producción masiva como juguetes, equipos de video, impresoras, electrodomésticos y son de amplio uso en la industria automotriz.

Dispositivo	Bits	ROM (Bytes)	RAM (Bytes)	FLASH/ (Bytes) OTP	Temporizadores	I/O	Puerto serial	A/D	Voltaje de operación (V)	Máxima frecuencia del bus (M)	Encapsulado
MC68HC05B6	8	6K	76	-	2 (16 Bits)	32	SCI	8 canales/8 Bits	3,3 - 5,5	4.0	56 SDIP
MC68HC05B32	8	32K	528	-	2 (16 Bits)	32	SCI	8 canales	3,3 - 5,5	2.1	56 SDIP
MC68HC705SR3	8	-	192	3.75K	1 (8 Bits)	32	-	4 canales	3,3 - 5,0	2.1	40 DIP
MC68HC705L16	8	-	512	16K	2 (16 Bits) 1 (8 Bits)	39	SIOP	-	3,3 - 5,0	2.1	80 QFP
XC68HC908GP20	8	-	512	20K FLASH	2 (16 Bits)	33	SCI/SPI	8 canales	3,0 - 5,0	8.0	40 DIP
MC68HC908GP32	8	-	512	32K FLASH	2 (16 Bits)	33	SCI/SPI	8 canales	3,0 - 5,0	8.0	40 DIP
MC68HC908JK1	8	-	128	1.5K FLASH	2 (16 Bits)	15	-	10 canales	3,0 - 5,0	8.0	20 DIP
MC68HC908JK3	8	-	128	4K FLASH	2 (16 Bits)	15	-	10 canales	3,0 - 5,0	8.0	20DIP
MC68HC12D0	8	-	192	-	1 (16 Bits)	16 Exp.	SCI/SPI	-	3,0 - 5,0	3	40 PDIP
MC68HC811E2	8	-	256	2048 EEPROM	1 (16 Bits)	38/22 Exp.	SCI/SPI	8 canales	5	2	48 DIP
XC68HC812A4	16	-	1K	4K EEPROM	8 (16 Bits)	+ de 91	SCI/SPI	8 canales	3,3 - 5	8/0	112 LQFP
XC68HC912B32	16	-	1K	768 EEPROM 32K FLASH	8 (16 Bits)	+ de 63	SCI/SPI	8 canales	5	8	80 QFP

Tabla 3.2 *Microcontroladores Motorola*

Existen cinco familias principales: la 68HC05, la 68HC08 y la 68HC11 de 8 *bits*, y la 68HC12 y la 68HC16 de 16 *bits*; cada una de ellas con una CPU diferente.

Dentro de la familia 68HC05 existen más de 30 microcontroladores distintos, todos construidos a partir de la misma CPU de 8 *bits* e incluyen RAM, ROM, puertos I/O y temporizadores; algunos tienen además, puertos seriales, convertidores análogo-digitales y memorias EEPROM o EPROM como lo describe la **tabla 3.2**.

Los microcontroladores de la familia HC08 poseen puertos seriales, conversión A/D, interfaces para redes CAN, PLL, PWM, etc., dependiendo de la referencia.

Microchip

Las raíces de los microcontroladores de esta empresa, se hallan en la Universidad de Harvard cuando se desarrollaba un proyecto para el Departamento de Defensa de E.E.U.U. en el cual se utilizaba un nuevo diseño en cuanto a la disposición e interacción de la CPU con la memoria, el cual tomó el nombre de arquitectura Harvard. Este tipo de arquitectura fue tomada por General Instruments para su CPU CP1600, y de esta empresa surgió, alrededor de 1985, una división que se ubicó en Arizona y que tomó el nombre de Microchip Technology, con el microcontrolador PIC como su producto principal.

Los microcontroladores PIC se clasifican en tres grupos dependiendo de la longitud de la palabra de instrucción que pueden manejar (12, 14 ó 16 *bits*), tomando las referencias 12CXXX, 16XXXX, 17XXX y 18XXX.

Nota: Las X en estos nombres quieren decir que en esos espacios se ubican diversos números y letras, según el tipo de integrado.

Sus fabricantes definen a los PIC como microcontroladores de 8 bits tipo RISC (*Reduced Instruction Set Computer* ó **Computadora con un conjunto reducido de instrucciones**) de bajo costo, bajo consumo de potencia y alta velocidad de operación.

No están muy alejados de la realidad: ya están en desarrollo microcontroladores PIC de hasta 40 MHz y el consumo de potencia es de 50

	TIEMPO	MEMORIA					PERIFERICOS											FUENTES
	Máxima frecuencia de operación (MHz)	Memoria de programa EPROM	Memoria de programa EEPROM	Memoria de datos(Bytes)	EEPROM de datos (Bytes)	Módulo temporizador	Capture/Compare/PWM Módulo	Puerto serial (SPI/I²C, SCI)	Puerto paralelo	Conversor análogo a digital (8 bit)	Conversor análogo a digital (10 bit)	Comparador	Referencia de voltaje interno	Fuentes de interrupción	Pines I/O	Rango de voltaje (Voltios)	Brown out	Encapsulado
PIC16C54	20	512	-	25	-	TMR0	-	-	-	-	-	-	-	-	12	2.5-6.25	-	18-pin DIP, 18-pin SOIC, 20 pin SSOP
PIC16C56	20	1K	-	25	-	TMR0	-	-	-	-	-	-	-	-	12	2.5-6.25	-	18-pin DIP, 18-pin SOIC, 20 pin SSOP
PIC16C61	20	1K	-	36	-	TMR0	-	-	-	-	-	-	-	3	13	3.0-6.0	-	18-pin DIP, 18-pin SOIC
PIC16C71	20	1K	-	36	-	TMR0	-	-	-	4 ch	-	-	-	4	13	3.0-6.0	-	18-pin DIP, 18-pin SOIC
PIC16C74	20	4K	-	192	-	TMR0, TMR1, TMR2	2	SPI/I²C/SCI	Si	8 ch	-	-	-	12	33	3.0-6.0	-	40-pin DIP, 44-pin PLCC, 44 pin QFP
PIC16C84	10	-	1K	36	64	TMR0	-	-	-	-	-	-	-	4	13	2.0-6.0	-	18-pin DIP, 18-pin SOIC
PIC17C42	25	2K	-	232	-	TMR0, TMR1, TMR2, TMR3	2	Si	-	-	-	-	-	11	33	4.5-5.5	-	40-pin DIP, 44-pin PLCC, 44 pin QFP
PIC17C44	25	8K	-	454	-	TMR0, TMR1, TMR2, TMR3	2	Si	-	-	-	-	-	11	33	2.5-6.0	-	40-pin DIP, 44-pin PLCC, 44 pin QFP
PIC16F873	20	-	4K	192	128	TMR0, TMR1, TMR2	2	MSSP Usart	No	-	5 ch	Si	-	13	22	-	Si	28-pin DIP SOIC
PIC16F877	20	-	8K	369	256	TMR0, TMR1, TMR2	2	MSSP Usart	Si	-	8 ch	Si	-	14	33	-	Si	40-pin DIP SOIC

Tabla 3.3 *Algunos modelos de microcontroladores PIC*

microwatios a 32 Khz de operación y 6 microwatios en modo de reposo o *standby*.

La arquitectura de los PIC se basa en registros con memorias y buses separados para las instrucciones y los datos, llamada arquitectura *Harvard*. La memoria y el bus de datos (RAM) son de 8 *bits* de ancho, mientras que la memoria (EPROM) y su bus tienen 12 *bits*.

La tecnología de los microcontroladores ha permanecido en constante evolución en cuanto a tecnología de fabricación y cualidades ofrecidas. Los microcontroladores PIC de última generación son máquinas asombrosas; por ejemplo se tienen proyectados PIC de tan solo 8 pines con conversor A/D y memoria *Flash*. Algunos llegan a poseer hasta 100 pines, y según la referencia, incluyen una buena variedad de módulos internos que cumplen con una gran cantidad de funciones como puertos seriales, PWM, comparadores, etc. En la **tabla 3.3** tenemos una lista de algunos modelos de estos microcontroladores con algunas de sus principales características.

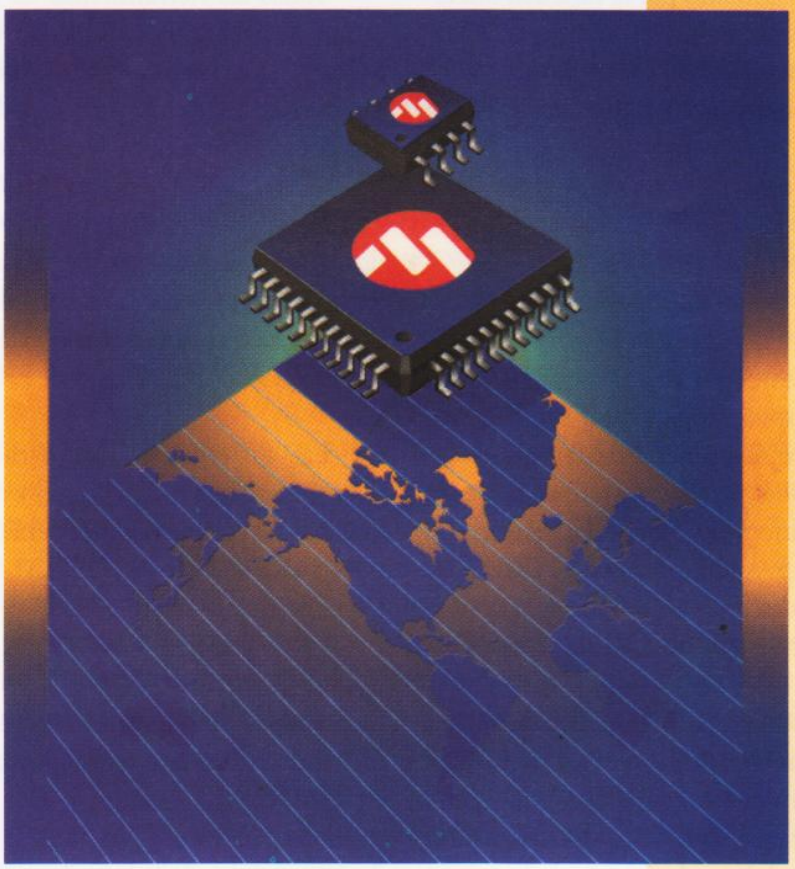
ATMEL

Esta empresa maneja tres grandes grupos de microcontroladores RISC cuyas CPU llegan hasta los 32 *bits*. El primer grupo tiene una arquitectura basada en el 8051 de Intel con memoria de programa tipo *Flash*. El segundo grupo son los microcontroladores AT91, los cuales soportan compilador en C, emulador, etc. El último grupo lo conforman los microcontroladores AVR, de arquitectura RISC y CPU de 8 *bits*, y módulos USART, SPI, ADC, etc. En la **tabla 3.4** se muestran las características principales de algunos modelos de esta marca.

	Memoria Flash	RAM	Temporizadores	Usart	Pines I/O	Vcc	Máxima frecuencia de oscilación (MHz)
AT91M40800	-	8K	3	2	81	1.8-3.6	40
AT91F40816	-	8K	3	2	85	2.7-3.6	40
AT91M40400	-	4K	3	2	32	2.7-3.6	33
AT89C51	4K	128	22	1	32	4.0-6.0	24
AT89C1051	1K	64	1	-	15	2.7-6.0	24
AT89C2051	2K	128	2	1	15	2.7-6.0	24

Tabla 3.4 *Microcontroladores ATMEL*

LECCIÓN 4



LOS MICROCONTROLADORES PIC

La familia de microcontroladores PIC ha conquistado, en los últimos años, un espacio importante entre los diseñadores de toda clase de circuitos de instrumentación, medición y control, así como entre los experimentadores y estudiantes de esta tecnología, por lo tanto, vamos a dedicar parte de este curso a explicarlos.

El crecimiento de la participación de estos microcontroladores en el mercado mundial ha sido posible debido a la capacidad de estos dispositivos de satisfacer las necesidades, cada vez más exigentes, de los ingenieros y técnicos de todo el mundo. Este crecimiento también se alimentó por la variedad ofrecida en sus presentaciones y tipos de memoria. Algunas de las razones que justificaron esta acogida son:

- Sencillez de manejo por su juego reducido de instrucciones
- Buena información: notas técnicas, proyectos en internet, etc.
- Precio y disponibilidad en el mercado
- Herramientas de desarrollo fáciles y baratas

La empresa Microchip fabrica, entre otros muchos productos, microcontroladores de 8*bits* de propósito general de arquitectura RISC (conjunto reducido de instrucciones), bajo consumo de potencia y una distribución interna en donde la memoria de programa (ROM) y la memoria de datos (RAM) se encuentran en bloques separados, cada una con su propio bus de acceso (estructura *Harvard*).

Ellos se clasifican dependiendo de la longitud de la palabra de sus instrucciones. Actualmente las tres familias de microcontroladores PIC son:

1. Línea Base o básica: palabras de 12 *bits* de longitud
2. Rango Medio: palabras de 14 *bits* de longitud
3. Rango Alto: palabras de 16 *bits* de longitud

Los dispositivos de la línea básica tienen las referencias 12CXXX, los de rango medio 16CXXX y los más avanzados se nombran como 17CXXX y 18CXXX; **figura 4.1**. Cuando desarrollamos un proyecto con microcontroladores PIC, debemos elegir el microcontrolador más adecuado dependiendo de sus características. Estas se relacionan directamente con las funciones que se puedan ejecutar con cada microcontrolador y su tecnología de fabricación.

Herramientas de desarrollo

Microchip y otras empresas, ofrecen un amplio rango de herramientas de desarrollo que permiten agilizar la elaboración y depuración de los programas. Estas pueden dividirse en cuatro categorías:

1. Generación de código
2. Depuración de los programas
3. Dispositivos programadores
4. Tarjetas de evaluación de los productos

Todas las herramientas de desarrollo de Microchip operan bajo el entorno del programa MPLAB. Las herramientas de generación de código son MPASM, MPLAB-C y MP-DriveWay™. Las herramientas para depuración del código son: PICMASTER® emulador In-Circuit, ICEPICt Emulador In-Circuit y el MPLAB-SIM o simulador de programas. Después de generado el código de la aplicación, el dispositivo debe ser programado. Microchip ofrece dos niveles de programadores: el PICSTART Plus y el PROMATE II.

Las tarjetas de demostración permiten al usuario evaluar la capacidad y disponibilidad del dispositivo para una aplicación determinada. Ellas son: PICDEM-1, PICDEM-2, PICDEM-3 y PICDEM-14A.

Arquitectura de los PIC

Podemos estudiar cada microcontrolador PIC teniendo en cuenta tres aspectos fundamentales:

1. El núcleo
2. Los periféricos
3. Las características especiales

El núcleo

Se refiere a las características fundamentales que se requieren para realizar su operación básica. Entre ellas están:

El circuito oscilador. Es un circuito que se utiliza para generar la señal de reloj del dispositivo. Esta señal es necesaria para que el microcontrolador ejecute las instrucciones y para que funcionen los periféricos. Entre los circuitos de oscilación más comunes están los basados en cristal, en red RC y los de resonador.

El sistema para reanudar (*Reset*). Se usa para llevar al microcontrolador a un estado conocido. En este estado se establecen por lo general las condi-

ciones de inicio con que siempre empieza el sistema y con las que un producto electrónico garantiza un buen funcionamiento de todas sus tareas.

La unidad central de proceso (CPU). Es la responsable de tomar la información de la memoria de programa (instrucciones) para controlar la operación de todo el sistema.

La unidad aritmético lógica (ALU). En un sistema con microcontrolador muchas de las instrucciones operan sobre la memoria de datos. Para operar en esta memoria, se requiere de la ALU, la cual permite el desarrollo de las operaciones artiméticas y lógicas sobre ella.

Organización del mapa de memoria del dispositivo. Existen dos bloques de memoria que se tratan en este punto: la memoria de programa, la cual se divide en bancos seleccionables por programa, y la memoria de datos; cada bloque cuenta con su propio bus. La secuencia de las instrucciones está controlada por el contador de programa (PC), el cual se incrementa en cada paso, para ejecutar las instrucciones grabadas en la ROM.

Operación de las interrupciones. Los microcontroladores PIC tienen varias fuentes de interrupción. Estas fuentes incluyen a los módulos periféricos.

Conjunto de instrucciones. Cada instrucción se divide en un código de operación (OPCODE), que especifica el tipo de instrucción, y uno o más operandos que especifican la operación de la instrucción. En el siguiente ejemplo observamos el código de operación de la instrucción MOVLW 06H:

13 ... 8	7 ... 0	(*bits*)
OPCODE	k (literal)	
MOVLW 30	06H 06	instrucción código operativo

Los periféricos

Los periféricos son los elementos que en realidad diferencian a los microcontroladores PIC de un microprocesador. Ellos hacen posible la comunicación con el mundo exterior (puertos I/O, manejadores LCD, entradas A/D, PWM, etc.) y la realización de diversas tareas tales como tener diferentes bases de tiempo. Los principales periféricos de un microcontrolador PIC son:

Las líneas o pines de entrada/salida (I/O) de propósito general. Permiten que el PIC reciba y envíe señales desde y hacia el mundo exterior. Además, se utilizan para vigilar y controlar otros dispositivos. Para darle más flexibilidad al dispositivo, algunos pines se multiplexan con funciones alternas. Esto quiere decir que en unos casos las mismas líneas pueden ser pines de entrada/salida o en otros casos, realizar otra función.

Los temporizadores. Se utilizan para establecer bases de tiempo confiables, medir el tiempo entre eventos, contar eventos externos, etc. En los microcontroladores PIC pueden existir diferentes tipos de temporizadores como:
Timer0. Temporizador de 8 *bits*.
Timer1. Temporizador de 16 *bits*.
Timer2. Temporizador de 8 *bits*.

Los módulos de captura, comparación y PWM (CCP). Cada módulo CCP contiene un registro de 16 *bits* que opera como registro de captura, como un registro de comparación o como un registro de PWM de 10 *bits*. Estos módulos son muy útiles en distintas aplicaciones tales como el control de motores o cargas resistivas, comparación de señales y otras.

Puerto serial síncrono (SSP). Este módulo es una interfaz serial para la comunicación con otros microcontroladores u otros periféricos tales como memorias seriales, registros de desplazamiento, conversores A/D, etc.

Puerto serial síncrono básico (BSSP). Es una interfaz serial para comunicarse con otros disposi-

tivos. Puede operar en dos modos distintos: SPI™ ó I 2 C™.

Puerto serial síncrono maestro (MSSP). Este es otro módulo para comunicación serial tipo SSP.

USART (SCI). Se denomina módulo universal receptor transmisor asíncrono. Es uno de los dos tipos de puertos seriales (el otro es el SSP). Puede configurarse en modo *full duplex* o como *half duplex*.

Referencias de voltaje. Este módulo se usa normalmente en conjunto con el módulo comparador. Se puede entender como una red de resistencias que proporciona un voltaje de referencia seleccionable.

Comparadores. Algunos PIC contienen dos comparadores análogos de señales. Las entradas a los comparadores se multiplexan con pines I/O.

Conversores A/D de 8 *bits*. Convierten una señal análoga en un dato digital de 8 *bits*.

Conversores A/D de 10 *bits*. Convierten una señal análoga a un dato digital de 10 *bits*.

Manejadores de *displays* o pantallas de cristal líquido (LCD). Este módulo genera el control de tiempos para manejar un panel LCD. También proporciona el control para los pixeles de datos del LCD.

Puerto paralelo esclavo (PSP). Algunos PIC poseen un puerto paralelo de 8 *bits*.

Características especiales

Las características especiales ayudan a disminuir el costo del sistema y a incrementar su funcionalidad y su flexibilidad. Los microcontroladores PIC de rango medio ofrecen algunas características que cumplen con esos objetivos:

Bits de configuración del dispositivo. Permiten que cada usuario personalice ciertos aspectos del microcontrolador que va a utilizar para su aplicación. Cuando el circuito se alimenta, el estado de estos *bits* determinan su modo de trabajo.

Power-on Reset (POR). Cuando se alimenta el sistema, se puede generar un estado de *reset* que lo obliga a llevar el contador de programa a la posición inicial.

Brown-out Reset (BOR). Origina un estado de *reset* en el momento en que se detecta una falla en el voltaje de alimentación.

Temporizador *Watchdog*. Es un oscilador tipo RC ubicado en el interior del circuito integrado que no requiere componentes externos. Se utiliza para prevenir fallas catastróficas en la operación de un sistema, cuando el PIC se bloquea por alguna razón. También se puede utilizar para temporizar o como detector de pulsos perdidos.

Modo de bajo consumo de potencia (*Sleep*). En este modo de trabajo el oscilador del PIC es desactivado y se presenta un consumo mínimo de potencia. Cuando el sistema lo requiere, el microcontrolador vuelve a su estado de consumo normal.

Oscilador interno RC. Cuando no se tienen requerimientos exigentes de manejo en la precisión del tiempo, se puede utilizar una red RC como oscilador externo lo que permite un diseño muy económico.

Programación serial dentro del circuito (*In-Circuit*). Es la posibilidad de programar los microcontroladores dentro de la aplicación o circuito final. Esto se hace mediante cinco líneas: una para el reloj, otra para los datos y otras tres líneas para la alimentación y el voltaje de programación.

Con todos estos factores, podemos establecer un diagrama de bloques genérico de un microcontrolador PIC, como el que se muestra en la **figura 4.1.**

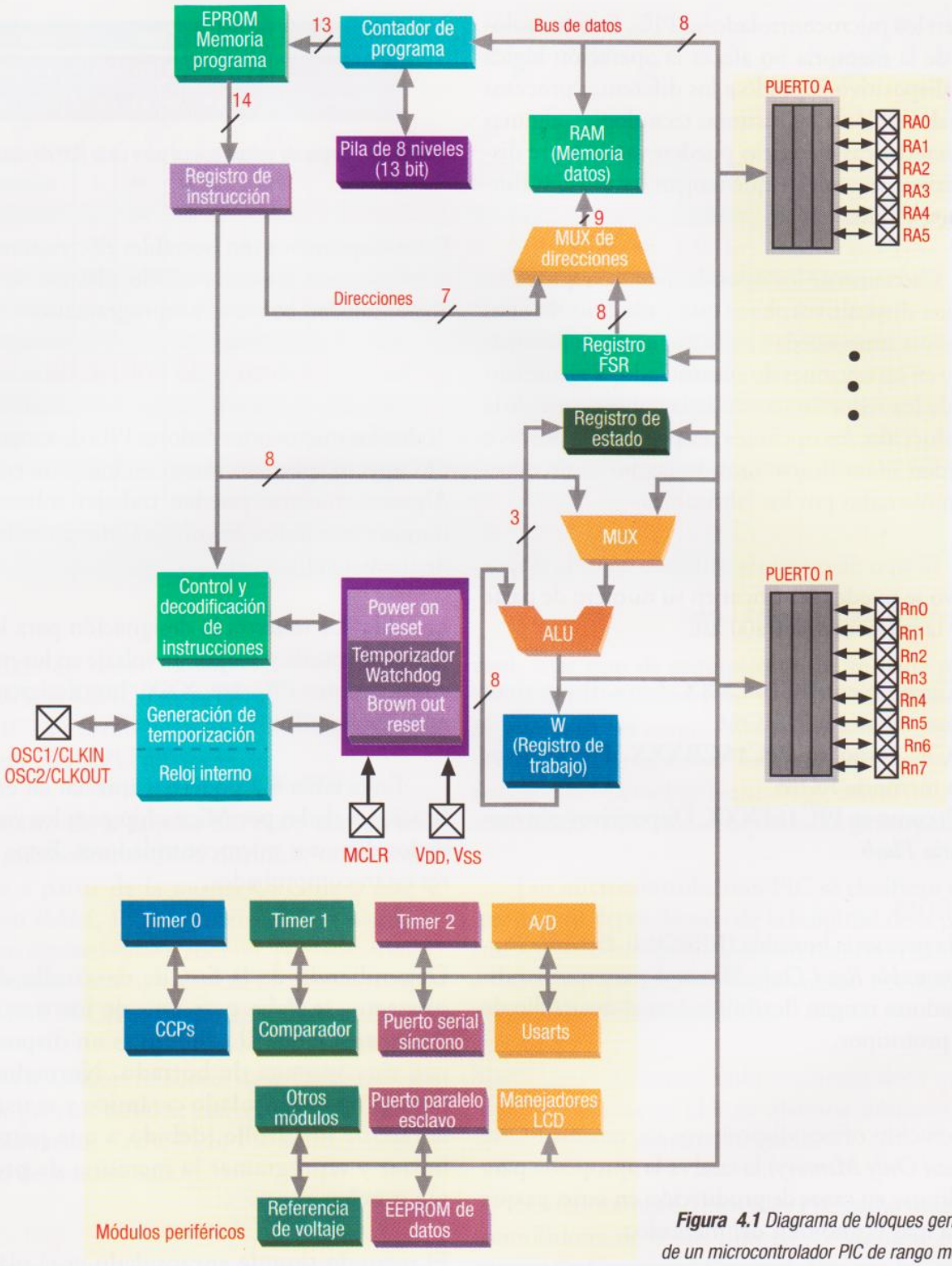


Figura 4.1 *Diagrama de bloques general de un microcontrolador PIC de rango medio*

Variedad de los dispositivos

Una vez se han analizado los requerimientos funcionales del dispositivo, se deben estudiar otros aspectos como: tecnología de la memoria, voltaje de operación, rango de temperatura de operación, frecuencia de operación y encapsulado. Microchip posee una buena cantidad de opciones entre las cuales podemos escoger la más indicada.

Variedades de memoria

Dependiendo de la aplicación y el modo de programación, hay varias tecnologías para la memo-

ria en los microcontroladores PIC. Esta tecnología de la memoria no afecta la operación lógica del dispositivo. Debido a los diferentes procesos de fabricación y las distintas tecnologías, algunas características eléctricas pueden variar entre dispositivos, a pesar de que tengan las mismas funciones y pines.

Cada uno de los tipos de memoria que están en los dispositivos, hace que cada uno de ellos presente una variedad en la respuesta de frecuencia y en las opciones de encapsulado. Dependiendo de los requerimientos de la aplicación y de la producción, las opciones propias del dispositivo pueden identificarse usando las hojas de datos suministradas por los fabricantes.

El tipo de memoria utilizado en cada dispositivo se puede identificar en su número de parte por la primera(s) letra(s) así:

1. **C**, como en **PIC16CXXX**. Estos dispositivos tienen memoria EPROM
2. **CR**, como en **PIC16CRXXX**. Dispositivos con memoria ROM
3. **F**, como en **PIC16FXXX**. Dispositivos con memoria *Flash*

EPROM

Es la memoria borrable (EPROM: *Erasable Programmable Read Only Memory*) para que los diseñadores tengan flexibilidad en el desarrollo de los prototipos.

ROM

Microchip ofrece dispositivos en versión ROM (*Read Only Memory*) la cual es la apropiada para utilizarse en casos de producción en serie, ya que es la más económica del mercado.

Tipo de memoria	Rango de voltaje	
	Estándar	Extendido
EPROM	PIC16CXXX	PIC16LCXXX
ROM	PIC16CRXXX	PIC16LCRXXX
FLASH	PIC16FXXX	PIC16LFXXX

Tabla 4.1 *Tipos de memorias y rangos de voltaje*

Rango de voltaje	EPROM		ROM		FLASH	
Estándar	C	4.5 - 6.0V	CR	4.5 - 6.0V	F	4.5 - 6.0V
Extendido	LC	2.5 - 6.0	LCR	2.5 - 6.0	LF	2.0 - 6.0

Tabla 4.2 *Rangos de voltaje típicos para cada tipo de dispositivo*

Memoria *Flash*

Estos dispositivos son borrables eléctricamente y se ofrecen en un encapsulado plástico de bajo costo; pueden borrarse y reprogramarse.

Opciones de rango de voltaje de operación

Todos los microcontroladores PIC de rango medio operan sobre un rango estándar de voltaje. Algunos modelos pueden trabajar sobre unos rangos extendidos de voltaje (y rango reducido de frecuencia).

La **tabla 4.1** muestra la designación para los tipos de memoria y rangos de voltaje en los microcontroladores PIC 16CXXX. Los designadores están en negrilla.

En la **tabla 4.2** podemos apreciar las especificaciones dadas por Microchip para los voltajes de los distintos microcontroladores. Estos valores están comprobados.

Variedades de encapsulado

Dependiendo de la fase de desarrollo de un proyecto, se debe usar uno de los tres tipos de encapsulado. El primero es un dispositivo con una ventana de borrado. Normalmente poseen un encapsulado cerámico y se usan en la fase de desarrollo, debido a que permiten borrar y reprogramar la memoria de programa muchas veces.

El segundo tipo de encapsulado es el plástico de bajo costo. Por último están los dispositivos DIE (dado) los cuales se caracterizan por no tener encapsulado. Se utilizan en aplicaciones de muy bajo costo donde el espacio de las tarjetas deba ser mínimo. En la **figura 4.2** se muestran los tres tipos de encapsulados.

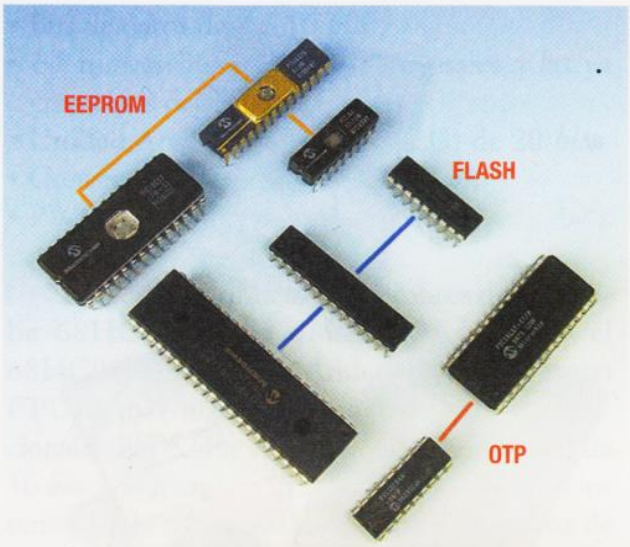


Figura 4.3 *Principales encapsulados de los microcontroladores PIC*

Dispositivos borrables con luz ultravioleta

La versión borrable UV de dispositivos de memoria de programa EPROM es óptima para el desarrollo de prototipos y programas piloto. Estos dispositivos pueden programarse una y otra vez. La cantidad de tiempo requerida para borrar completamente un dispositivo UV depende de la longitud de onda del haz, su intensidad, distancia entre la fuente y el microcontrolador, y de la tecnología del dispositivo (qué tan pequeñas son las celdas de memoria).

Nota: las bombillas fluorescentes y la luz del sol, emiten luz ultravioleta con una longitud de onda que permite el borrado. Si en un dispositivo de este tipo no se protege la ventana, puede ocurrir, con el tiempo, que las celdas de la memoria se borren. El tiempo de borrado con una luz fluorescente es aproximadamente de tres años, mientras que con la luz del sol se requiere de aproximadamente una semana. Para prevenir la pérdida de los datos, se debe colocar una superficie opaca (p. ej. cinta aislante) sobre la ventana de borrado.

Dispositivos programables una vez (One-Time-Programmable: OTP)

Los dispositivos OTP, se encapsulan en plástico y permiten que el usuario los programe una sola vez. Si el programa se daña o se tiene que cambiar, hay que programar e instalar en el circuito un nuevo dispositivo.

Dispositivos FLASH

Un microcontrolador tipo FLASH permite modificar el contenido de su memoria mediante una carga eléctrica. Esto significa que el sistema puede ser diseñado para programación dentro del circuito. Como no requieren de una ventana de borrado, estos dispositivos tienen encapsulado plástico, lo que los hace muy económicos.

Dispositivos EEPROM

Un dispositivo EEPROM permite que su memoria sea borrada mediante una carga eléctrica. Esto significa que el sistema puede diseñarse para que el borrado y el reprogramado puedan hacerse dentro del circuito. Tampoco requieren ventana de borrado y se encapsulan en plástico.

Dispositivos ROM

Los dispositivos ROM tienen su memoria de programa establecida desde el momento de su fabricación. Como ella no puede cambiarse, estos dispositivos tienen la presentación en plástico más económica del mercado.

DIE

Esta opción permite el desarrollo de tarjetas muy pequeñas. La fabricación con DIE requiere equipos y conocimientos especiales, lo que implica que el número de casas fabricantes que soportan esta tecnología sea limitado.

Servicios especiales

Para algunos usuarios que incorporen la tecnología OTP, debido a un código fijo para sus productos, Microchip ofrece dos encapsulados especiales: la programación rápida para producción y la programación serial rápida para producción. Ellas les permiten a estos usuarios ahorrar mucho tiempo en el proceso de fabricación de sus productos.

a. **Programación rápida para producción** (QTP: *Quick Turn Production*)

Microchip ofrece este servicio de programación para ordenes grandes de producción de empresas o usuarios, que no posean un medio masivo de programación de MCU. Estos dispositivos son idénticos a los OTP pero con todas las posiciones de la EPROM, y las opciones de configuración ya programadas.

b. **Programación serial rápida para producción** (SQTP: *Serialized Quick Turn Production*) Microchip ofrece este servicio de programación en el cual a cada dispositivo se le programa un número serial único. Estos números seriales pueden ser aleatorios, pseudoaleatorios o secuenciales. Este tipo de programación se hace para que cada dispositivo tenga un número único que pueda utilizarse como código de entrada, *password* o número de identificación.

PROGRAMADORES Y ENTRENADORES (*Trainers*)

Los programadores son dispositivos que permiten grabar en la memoria del microcontrolador el programa escrito y ensamblado en la computadora. Algunos programadores trabajan conectándose al puerto paralelo como el K-220 de CEKIT, **figura** 4.4; y otros, conectándose al puerto serial como los que se muestran en la **figura** 4.5.

Figura 4.5 *Programadores por el puerto serial*
a) Programador K-224 de CEKIT
b) PICSTART Plus de Microchip
c) Programador Clear View Mathias

Un entrenador es un dispositivo que permite realizar una serie de prácticas con microcontroladores. Se compone de una serie de módulos independientes que se pueden interconectar entre sí y con los cuales se facilita el aprendizaje de las prácticas más comunes y fundamentales que se deben asimilar en esta tecnología. El K-148 de CEKIT, **figura** 4.7, es un entrenador que tiene módulos como el de manejo de *displays* de siete segmentos, la adquisición de variables análogas y comunicaciones seriales, entre otros.

Figura 4.4 *Programador por puerto paralelo, K-220 de CEKIT*

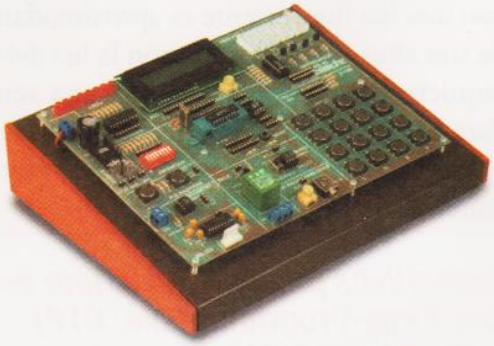
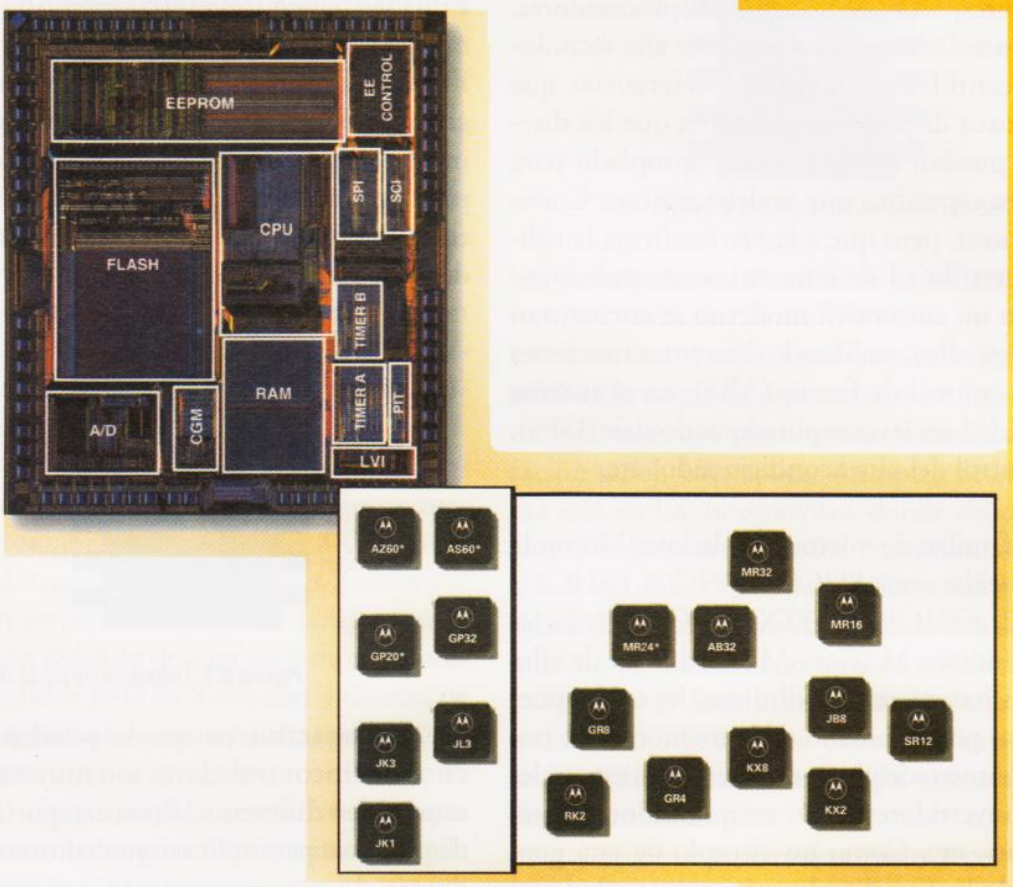


Figura 4.6 *Entrenador para microcontroladores PIC, referencia K-148 de CEKIT*

LECCIÓN 5



LOS MICROCONTROLADORES MOTOROLA

Esta familia de microcontroladores es una de las más difundidas y utilizadas en todo el mundo. Sus aplicaciones incluyen las comunicaciones, el control, la electrónica automotriz, etc. En esta lección haremos una introducción a sus aspectos más importantes como las familias en que están divididos, sus características, sus ventajas y en general, daremos un rápido vistazo a esta tecnología.

Los microcontroladores Motorola tienen la reputación de ser muy confiables, fáciles de programar y, ante todo, ser unos de los más utilizados en todo el mundo. Y no es de extrañar ya que esta firma es líder en la fabricación de semiconductores, especialmente microprocesadores, microcontroladores y productos de alta tecnología. La cantidad de familias y referencias que poseen estos dispositivos permiten que los diseñadores puedan escoger el más apropiado para cada tarea específica que se deba realizar. Como dato curioso, pero que a la vez confirma la calidad y versatilidad de estos microcontroladores, es que en un automóvil moderno se encuentran hasta 50 de ellos, realizando diferentes funciones como el control de frenos (ABS), en el sistema de seguridad, en las computadoras de viaje (GPS), en el control del aire acondicionado, etc.

Las familias de microcontroladores Motorola más conocidas son: 68HC05, 68HC08, 68HC11, 68HC12, 68HC16, 683XX, MPC5XX y la familia de núcleo M (Core M). Cada una de ellas tiene muchas referencias distintas, las cuales pueden variar por el tamaño de la memoria que poseen, el número de pines, características especiales como convertidores A/D, temporizadores, interrupciones, etc. Como un ejemplo de esta gran variedad, la familia de microcontroladores 68HC05 posee más de 100 miembros. Como el tema es tan amplio, en lecciones posteriores trataremos principalmente la familia 68HC08, que es la más indicada para iniciarnos en esta tecnología y haremos una breve descripción de las familias 68HC11 y 68HC12.

La familia HC05

Esta es una de las familias de microcontroladores más populares en el mundo, con más de 2 billones de unidades vendidas. Posee una ventaja adicional y es la compatibilidad de sus programas con la familia HC08, lo que permite hacer migración de diseños hacia dispositivos de más alto rendimiento de una manera fácil y rápida. Cuando se especifica una referencia en particular, ésta puede sufrir algunas variantes y aún así corresponder a un dispositivo 100% compatible. Por ejemplo, hay un microcontrolador 68HC705KJ1, otro 68HC805KJ1 y otro 68HC905KJ1. La diferencia radica en que el primero (705) tiene memoria de programa tipo EPROM, el segundo (805) memoria de programa tipo EEPROM y el tercero (905) memoria FLASH. Lo que sí queda claro es que existe una gran variedad de referencias para escoger; esto implica que el diseñador debe estudiar a fondo para decidirse por la más adecuada. En la **figura 5.1** se muestra el diagrama de pines de uno de estos microcontroladores.

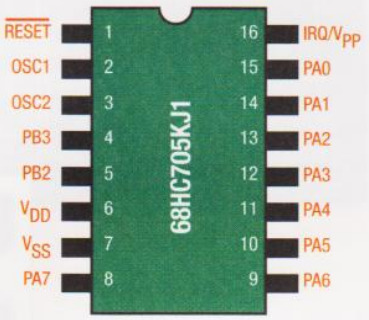


Figura 5.1 *Diagrama de pines del 68HC705KJ1*

Los aparatos en que se pueden encontrar estos microcontroladores son muy variados y en ramas bien diferentes. Estas categorías incluyen dispositivos para aplicaciones automotrices, productos de consumo, comunicaciones, computadoras y aplicaciones industriales, entre otras. A continuación haremos una breve descripción de algunas subfamilias o divisiones que tiene la familia HC05.

68HC05-B. Es un dispositivo de propósito general cuyas principales ventajas son un poderoso temporizador y una memoria para datos tipo EEPROM de 256 posiciones. Otras características incluyen memoria de programa desde 4K hasta 32K, memoria RAM desde 176 hasta 528 *bytes*, 8 canales A/D, comunicación serial no sincrónica, etc.

68HC05-C y 68HC05-D. Son elementos de propósito general cuya principal ventaja es su temporizador de 16 *bits*, el cual está acompañado por un módulo de captura y un módulo de

comparación. Además, ofrece tamaños de memoria de programa desde 4K hasta 16K, que lo capacitan para realizar tareas bastante complejas. Posee interfaz de comunicación no sincrónica SCI (*Serial Communications Interface*), con velocidad ajustable por programa desde 75 Hz hasta 131 KHz; también incluye una interfaz SPI (*Serial Periferical Interface*) de 4 hilos y alta velocidad. El *watchdog timer* o circuito de vigilancia también es una de sus características importantes.

68HC05-J y 68HC05-K. Estas son las familias de más bajo costo y vienen en encapsulados de 20 y 16 pines. El tamaño de su memoria de programa va desde 0,5K hasta 2K, con memoria RAM de 32 hasta 128 *bytes*. El microcontrolador 68HC705KJ1 es el más económico de todos.

68HC05-P. Es una familia de microcontroladores de propósito general de 28 pines, poseen memoria de programa desde 2K hasta 8K con memoria RAM desde 96 hasta 192 *bytes*. Tienen comunicación serial no sincrónica, temporizador de 16 *bits* con módulo de captura y de comparación, convertidor A/D de 8 *bits* con 4 canales de entrada y capacidad de manejar alta corriente en algunos pines.

68HC05-L. Su principal característica es que posee un controlador para manejar una pantalla de cristal líquido o LCD, con un tamaño que va desde 8 caracteres por 4 filas hasta 16 caracteres por 60 filas. La memoria de programa va desde 0,6K hasta 24K con memoria RAM de 32 hasta 768 *bytes*. Incluye temporizador de 16 *bits* con módulos de captura y comparación, comunicación serial sincrónica y no sincrónica.

La familia HC08

Estos microcontroladores permiten una fácil migración del código desde el HC05, poseen una velocidad interna hasta de 8 MHz, una pila de 16 *bits*, instrucciones de multiplicación y una memoria de 64K para programa y datos; su diseño minimiza el consumo de potencia. Hay versiones con memoria FLASH y EPROM. En la **figura 5.2** se muestra el diagrama de pines de uno de estos microcontroladores.

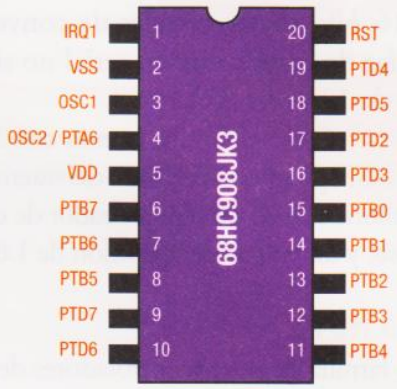


Figura 5.2 *Diagrama de pines del 68HC908JK3*

Las principales subfamilias de la HC08 son:

68HC08 AS. La principal característica de esta subfamilia es que incorpora un controlador de bus multiplexado que permite una avanzada comunicación serial. Una aplicación típica de este sistema está en los automóviles, donde muchos microcontroladores se pueden comunicar sobre un bus de uno o dos hilos para el diagnóstico de cada una de las operaciones del vehículo con un riesgo mínimo de interferencia o ruido en los datos.

68HC08 AZ. Esta subfamilia contiene una interfaz CAN. Ésta es un protocolo muy popular en redes de automatización industrial y redes automotrices.

68HC08 GP. Esta familia es muy recomendada para aplicaciones de propósito general, cuenta con memoria FLASH, convertidores A/D, puertos seriales sincrónicos y no sincrónicos, PLL, temporizadores y PWM.

68HC08 JL. Esta subfamilia brinda unas poderosas características y es especial para aplicaciones de propósito general donde se requieran diseños de bajo costo. Se encuentran disponibles en encapsulados de 20 y 28 pines, memoria FLASH o ROM, conversores A/D, entre otros. Las versiones JK, por ejemplo el 68HC908JK3, son parte de ella

68HC08 MP y MR. Estas subfamilias incorporan un sistema avanzado de PWM de 6 canales y 12 *bits*, convertidores A/D y puertos seriales, que las hace ideales para el control de motores trifásicos.

68HC908KX8/2. Esta subfamilia posee integrados con memoria FLASH de 2 hasta 8Kbytes, 196 *bytes* de memoria RAM, un temporizador de 2

canales y 16 *bits*, cuatro canales de conversión A/D, interfaz de comunicación serial no sincrónica y más de 13 pines I/O.

68HC908RK2. Esta subfamilia cuenta con 2K de memoria FLASH programable dentro del sistema, 128 *bytes* de memoria RAM, un temporizador de dos canales y 16 *bits*, y un voltaje de operación de 1.8 V.

La familia HC11

Esta es una familia de microcontroladores de 8 *bits* de muy alto rendimiento, su arquitectura es compatible con la HC05 y su programación es compatible con la familia de microcontroladores de 16 *bits* 68HC12; esto hace que la migración de diseños de una familia a otra sea bastante rápida. Dentro de sus características más sobresalientes se encuentran:

- Ha sido optimizada para un bajo consumo de potencia
- Posee un sistema de interrupciones muy avanzado
- Muchos de sus miembros poseen memoria EEPROM
- Los últimos miembros de esta familia incluyen un coprocesador matemático con Unidad Aritmética Lógica (ALU) de 16 *bits* para rápidas operaciones de multiplicación y división
- Un registro acumulador de 16 *bits*, o dos de 8 *bits*
- Seis modos de direccionamiento

La familia 68HC11 posee un miembro llamado 68HC11E9, el cual fue el primer microcontrolador en incorporar memorias EPROM y EEPROM en el mismo circuito integrado; además, incluye otras características como son 12K de memoria EPROM, 512 *bytes* de EEPROM, 512 *bytes* de RAM, interfaz SCI, convertidor A/D de 8 canales/8 *bits* y 38 pines de entrada/salida. Este microcontrolador es muy utilizado por los experimentadores en robótica. Dentro de los 68HC11 se tienen seis principales series de microcontroladores; todas cuentan con puertos seriales SCI y SPI, memoria EEPROM y ADC; ellas son:

68HC11 D. Esta serie ofrece una alternativa económica para aplicaciones donde no se requieran funciones avanzadas, ni mucha cantidad de memoria.

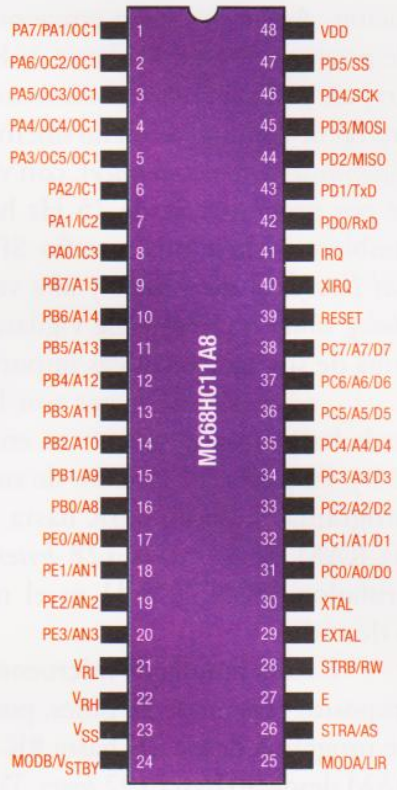


Figura 5.3 *Diagrama de pines del 68HC11*

68HC11 E. Fue la primera en combinar EEPROM y EPROM en un solo circuito integrado.

68HC11 F. Tienen buena capacidad de pines I/O, memoria RAM de 1K y un bus de 5MHz.

68HC11 K y KS. Son dispositivos muy poderosos; el 68HC11K4 y el 68HC11KW1 ofrecen alta velocidad, MMU (unidad para el manejo de memoria) y PWM.

68HC11 P. Esta serie ofrece PLL, buena memoria y tres puertos seriales. En la **figura 5.3** se muestra el diagrama de pines de el microcontrolador MC68HC11A8.

La familia HC12

Este es un poderoso microcontrolador de 16 *bits* y según Motorola reune lo mejor del 68HC11 y mucho más. Entre sus principales características se encuentran:

- Bajo consumo de potencia y operación a bajo voltaje
- Memorias *Flash* y EEPROM en un solo circuito integrado
- Instrucciones de lógica difusa (*Fuzzy Logic*)

- Bus de datos de 8 ó 16 *bits*
- 64 nuevas instrucciones adicionales a las ya conocidas del 68HC11
- Unidad Aritmética Lógica (ALU) de 20 *bits*
- Convertidor A/D de 8 canales/8 *bits*
- PWM

Dos de los principales miembros de la familia 68HC12 son el 68HC812A4 (A4) y el 68HC912B32 (B32). Ambos se basan en una CPU12, más una serie de módulos periféricos adicionales. El B32 fue el primer microcontrolador de 16 *bits* con memoria FLASH EEPROM en su interior. En la **figura 5.4** se muestra el diagrama de pines de uno de estos microcontroladores.

68HC812A4. Sus principales características son una CPU de 16 *bits*, 1 K de RAM, 4 K de EEPROM, dos interfaces seriales (SCI) y una interfaz serial periférica (SPI), un temporizador y conversores A/D.

68HC912B32. También posee una CPU de16 *bits*, 32 K de flash EEPROM, 1K de RAM, 768 *bytes* de EEPROM, interfaces SCI y SPI, un temporizador de 8 canales, conversores A/D, y PWM de cuatro canales.

HC912DG128. CPU de 16 *bits*, altamente compatible con el HC11, ALU de 20 *bits*, 128K de flash EEPROM, 2K de EEPROM, 8K de RAM, dos conversores A/D, dos temporizadores de 8 canales y 10 *bits* de resolución, filtro pasabajo, módulos CAN, cuatro canales PWM, módulo contador de 16 *bits*, interfaces seriales SCI, SPI, I2C, etc.

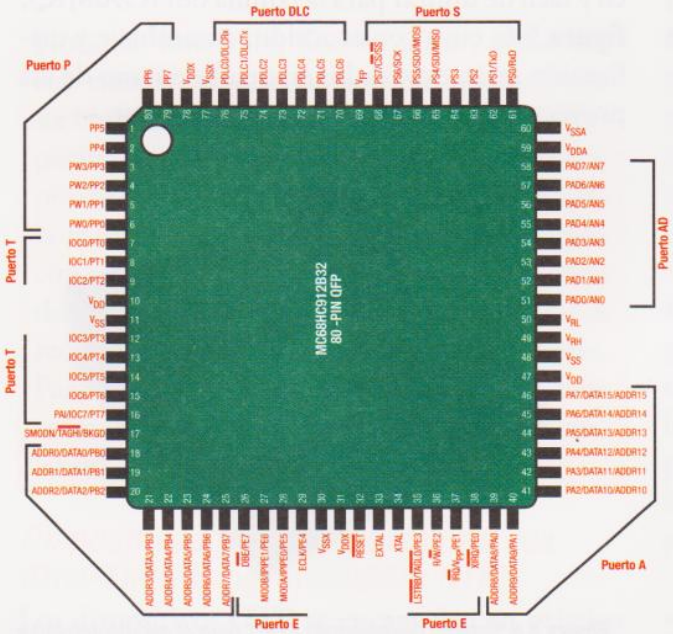


Figura 5.4 Diagrama de pines del 68HC912B32

La familia HC16

También se basa en una CPU de 16 *bits*. Su conjunto de instrucciones soporta lenguajes de alto nivel. La CPU de 16 *bits* es compatible con los microcontroladores HC11, algunas de sus principales características son:

- Funciones de procesamiento digital de señales (DSP)
- 1 Mbyte de espacio para memoria de programa
- 1 Mbyte de espacio para memoria de datos
- *2* acumuladores de propósito general de 16 *bits*
- 3 registros índice de 16 *bits*
- Rápida respuesta ante interrupciones
- Módulo CAN
- Versiones de 16, 20 y 25 MHz
- Convertidor A/D

Las principales variaciones de la familia HC16 son:

68HC16 Z. Es la más adecuada para aplicaciones con mucha memoria y que utilicen comunicaciones seriales y funciones análogas.

68HC16 Y. Esta familia integra en los circuitos memoria FLASH o ROM, unidad de procesamiento de tiempo, tres interfaces seriales (dos SCI y un SPI) y convertidores A/D.

68HC16 X y R. Son de bajo costo, memoria FLASH, módulos seriales y funciones análogas.

La familia NÚCLEO M (M CORE)

Estos poseen una arquitectura RISC de 32 *bits*, instrucciones de 16 *bits*, permiten la emulación gracias a su arquitectura interna y poseen un diseño para bajo consumo de potencia.

La familia MPC5XX

Tienen arquitectura de 32 *bits*, reloj a 40 MHz,

unidad para el trabajo de operaciones de punto flotante, interfaz para la emulación, etc.

La familia 683XX

Estos microcontroladores están diseñados para soportar lenguajes de alto nivel, cuentan con 16 Mbytes de memoria de programa, poseen un bus dinámico de 8 ó 16 *bits,* y vienen en versiones de 16MHz, 20MHZ ó 25 MHz. Las principales subfamilias de esta categoría son:

68331 y 32. Son subfamilias especiales para aplicaciones donde se requiere mucha memoria, cuentan con puertos seriales y módulos avanzados de temporización; la 68CK338 cuenta además con un reloj en tiempo real y un módulo temporizador de 20 canales. Las subfamilias 68336/68376 se componen de dispositivos diseñados para trabajar bajo condiciones atmosféricas complicadas, al igual que donde se requieran altas velocidades, comunicaciones seriales, redes CAN y sistemas sofisticados de adquisición desde múltiples fuentes.

Sistemas de desarrollo

Motorola y muchas empresas independientes de desarrollo electrónico, tienen disponible una gran variedad de herramientas para trabajar con estos dispositivos. Entre ellas se cuentan, emuladores, analizadores lógicos, programadores, tarjetas de evaluación, simuladores, compiladores en lenguaje C, sistemas operativos en tiempo real, ensambladores y depuradores. Todos estos elementos permiten desarrollar, monitorear, probar y depurar su código, para que las aplicaciones funcionen bien y rápido. Existe una gran cantidad de modelos, los cuales se ajustan a las necesidades y presupuestos disponibles, **figura** 5.5.

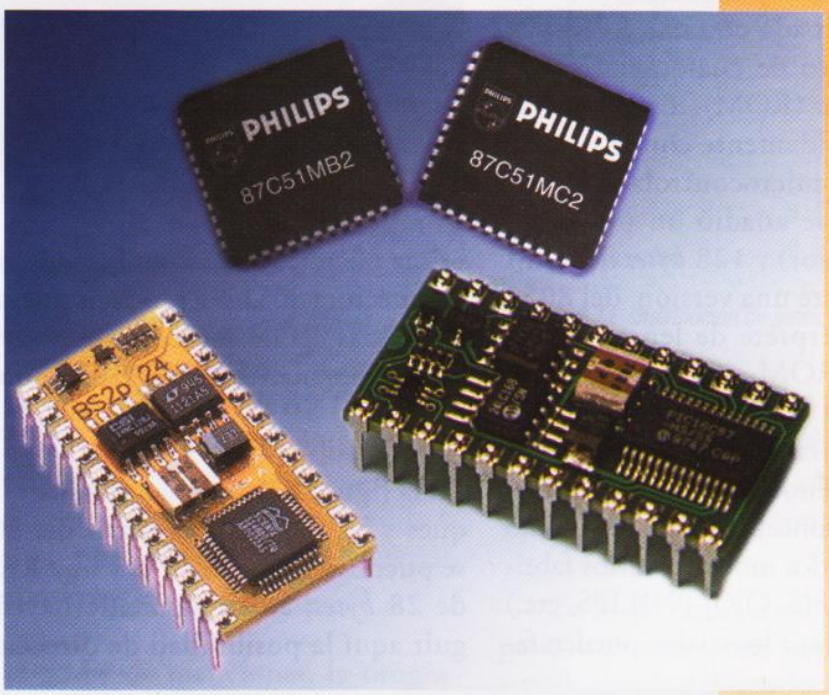
Algunos de estos dispositivos son de bajo costo, pero a pesar de ello ofrecen muchas ventajas, como por ejemplo, capacidad de emulación en el circuito de la aplicación y, aunque esta tarea no se ejecute en tiempo real, es de gran ayuda para agilizar el trabajo y disminuir el tiempo de desarrollo de los programas. Otras características son:

- Ambiente de desarrollo totalmente integrado bajo Windows
- Se puede examinar y modificar la memoria, los registros y los pines de entrada/salida
- Acepta hasta 64 *breakpoints* (puntos de parada)
- Diferentes modos de ejecución en pasos o de corrido
- Programa microcontroladores OTP y EPROM
- Conexión serial a la computadora

CEKIT S.A., como empresa líder en el diseño y fabricación de material didáctico, tiene disponible un sistema de desarrollo muy económico y fácil de utilizar para la familia 68HC908JK3, **figura** 5.6, cuya construcción, ensamblaje y utilización serán explicados en detalle en uno de los proyectos que forman parte de este curso.

Figura 5.5 *Sistema de desarrollo de Motorola para el 68HC05*

Figura 5.6 *Sistema de desarrollo CEKIT para el microcontrolador 68HC908JK3*



LOS MICROCONTROLADORES INTEL, ATMEL y BASIC Stamp

Daremos un vistazo rápido a estas tres marcas que también son ampliamente utilizadas, algunas por su potencia, otras por su facilidad de manejo. De cualquier manera, se constituyen en excelentes opciones para el desarrollo de productos electrónicos. El microcontrolador 8051 de INTEL y las versiones que de él se derivan, han obtenido en el medio industrial un éxito indiscutible, los ATMEL son muy utilizados en aplicaciones médicas y en redes de automatización, y los BASIC Stamp son muy apetecidos por la facilidad en su programación.

Microcontroladores INTEL

A comienzos de los años 80 Intel propuso el modelo de microcontrolador denominado 8051. Esta primera versión tenía tres variantes: el 8051 propiamente dicho, dotado de una memoria ROM interna programada en fábrica; el 8751, equipado con una EPROM; y el 8031, desprovisto de cualquier tipo de memoria permanente (ROM). Esta primera serie se completó rápidamente con la versión 8052. A este último microcontrolador, muy parecido al 8051, se le añadió un periférico (un tercer temporizador) y 128 *bytes* de RAM interna. También existe una versión del 8052 que contiene un intérprete de lenguaje BASIC en su memoria ROM.

Así nació la famosa familia 51, la cual se completó con otros miembros. Todos estos microcontroladores se han concebido bajo un mismo núcleo. Esta característica motivó a varios fabricantes (SIEMENS, MHS, OKI, PHILIPS, etc.) a firmar acuerdos gracias a los cuales pueden fabricar microcontroladores basados en la misma arquitectura 8051 o a producir nuevas versiones mejoradas del mismo circuito integrado, incorporándoles nuevos periféricos, **figura 6.1**.

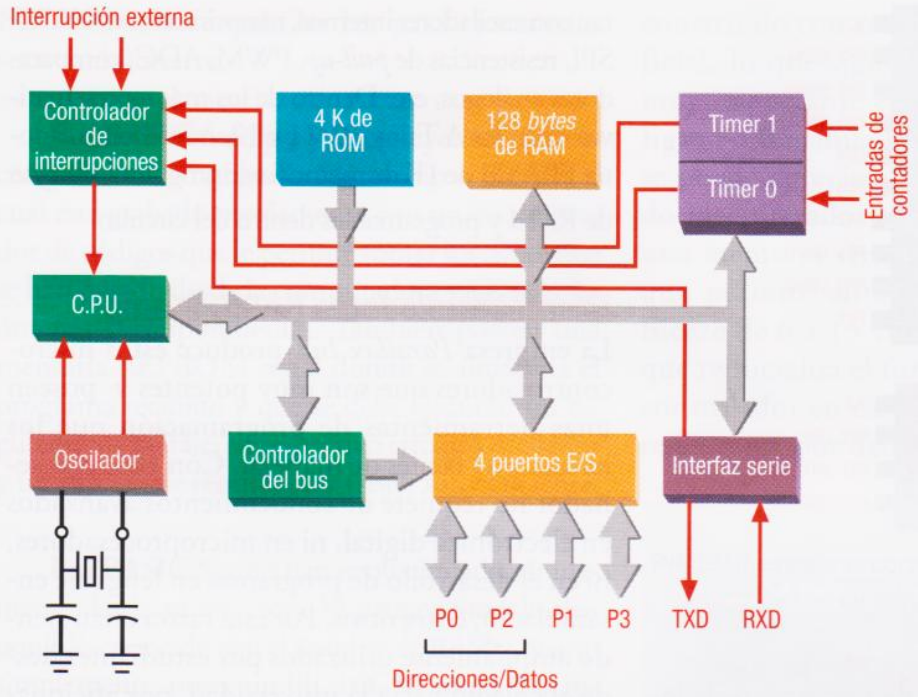
Figura 6.1 *Microcontroladores PHILIPS. Su núcleo es el 8051*

Núcleo 8051

En la **figura 6.2** se aprecia el diagrama de bloques y el diagrama de pines del 8051. En él se puede apreciar la CPU y una RAM interna de 28 *bytes*; adicionalmente hay que distinguir aquí la posibilidad de direccionamiento

VERSIÓN			MEMORIA								
Sin ROM	ROM	EEPROM	ROM	RAM	Timers 16 bits	Puertos 8 bits	Interrupciones/niveles	Puertos serie	Frecuencia en MHz	Contadores/temporizadores	Particularidades
Gama de microcontroladores MCS-51 de INTEL (1991)											
8031	8051	-	4 K	128	2	4	6/2	1	3.5 a 12		Versión NMOS
8032AH	8052AH	8752BH	8 K	256	3	4	8/2	1	3.5 a 12		
83C51FA	80C51FA	87C51FA	8 K	256	3+PCA	4	7/2	1			Posibilidad de salidas PWM, entradas con captura y WDT
83C451	80C451		4 K	128	2	7	5/2	1			7 puertos
Gama de microcontroladores SIEMENS (1991)											
80535	80515		8 K	256	3+WDT	6	1/4	1	1.2 a 12		8 entradas analógicas de 8 bits; 4 salidas PWM 4 entradas con captura; 4 niveles de prioridad de interrupciones
80C537	80C517		8K	256	3+WDT	7	14/4	2	1 a 16		12 entradas analógicas; 8 punteros de 16 *bits*; unidad aritmética lógica de 16 *bits*; 21 salidas PWM; 5 entradas con captura; 2 puertos serie; 4 temporizadores de 16 *bits*; WDT programable
80C17A	83C517A5		32 K	2.25K	4+2WDT	7	14/4	2	1 a 18		RAM interna de 2.25 Kb convertidor A/D de 10 *bits*
Gama de microcontroladores PHILIPS (1992)											
80C451	83C451	87C451	4 K	128		7		UART	12 y 16	2	
80C592	83C592	87C592	16 K	512		6		UART + CAN	16	3+WDT	CA/D 8E/10 *bits*; 2 salidas PWM e interfaz de bus CAN
80C528	83C528	87C528	32 K	512		4		UART + PC	16, 20 y 24	3+WDT	

Tabla 6.1 *Gama de microcontroladores 8051 y algunos derivados*



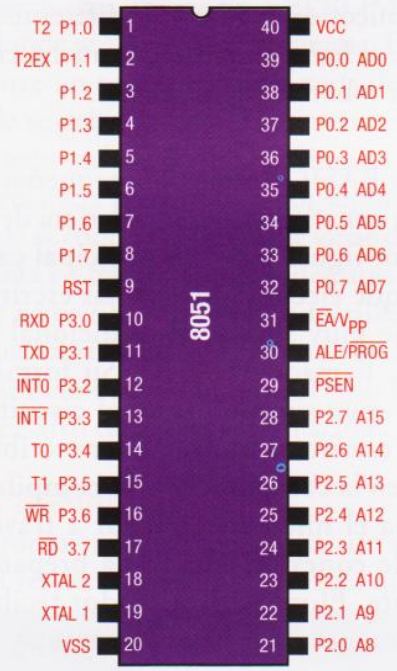


Figura 6.2 *Diagrama de bloques y de pines del 8051*

de memorias externas, en esos casos uno de sus puertos debe utilizarse en esa función, cambiando su característica como puerto a bus externo. El 8051 contiene, entre otros, un puerto serial y dos temporizadores de 16 bits. En la **tabla 1.1** se encuentran algunos de los modelos más representativos de esta familia.

Todos los fabricantes tienen disponibles los microcontroladores 8051, 8052 y sus versiones CMOS en encapsulado DIP de 40 pines o patillas. Algunos los ofrecen en otros encapsulados. Dentro de sus pines podemos destacar el pin $\overline{EA}$ el cual es una entrada que permite configurar el microcontrolador para sistemas con buses externos. Posee tres puertos: el puerto 0 cuyas líneas son de colector abierto, el puerto 1 quien posee resistencias internas de *pull-up*, el puerto 2 también con *pull-up* y el cual es utilizado durante el acceso a la memoria externa. Por último está el puerto 3 que posee funciones adicionales así:

- P3.0 RXD entrada de interfaz serie
- P3.1 TXD salida de interfaz serie
- P3.2 $\overline{INTO}$ entrada para interrupción externa
- P3.3 $\overline{INT1}$ entrada para interrupción externa
- P3.4 T0 entrada de contador para el temporizador 0
- P3.5 T1 entrada de contador para el temporizador 1
- P3.6 $\overline{WR}$ salida de escritura de la memoria externa
- $\overline{RD}$ salida de lectura de la memoria externa

En una lección posterior explicaremos en detalle este microcontrolador.

Microcontroladores ATMEL

Los microcontroladores ATMEL, **figura 6.3**, han tenido una gran acogida en el mercado; de ellos se pueden destacar tres familias, la primera está basada en el 8051, la segunda es la AT91 ARM Thumb y por último, los AVR. Veamos ahora un poco de cada uno de estos grupos.

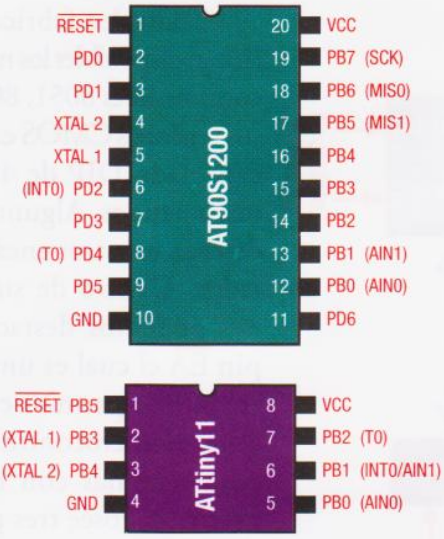


Figura 6.3 *Microcontroladores ATMEL AVR*

Basados en arquitectura 8051

ATMEL Ofrece una línea de microcontroladores basados en la arquitectura 8051, con memoria de programa FLASH dentro del circuito integrado, al igual que versiones OTP.

AT91 ARM Thumb

Los microcontroladores AT-91 son microcontroladores de 32 *bits* lo que permite el desarrollo de protocolos de comunicación, compresión de datos y control. Cuando se construye un producto electrónico alrededor de un núcleo ARM se garantiza la durabilidad y calidad del mismo.

Los microcontroladores AT91 son muy apetecidos para aplicaciones de bajo consumo de potencia y otras en tiempo real; se han hecho aplicaciones muy exitosas en automatización industrial, dispositivos MP3, productos para la adquisición de datos, equipos médicos, GPS y sistemas de redes.

La serie AT91 cuenta con un amplio conjunto de herramientas de programación de alto nivel, como compiladores de lenguaje C, emuladores y sistemas operativos en tiempo real.

Microcontroladores AVR

Son microcontroladores de 8 *bits* cuyas instrucciones se ejecutan en un solo ciclo de máquina. Cuentan con osciladores internos, temporizadores, UART, SPI, resistencias de *pull-up*, PWM, ADC, comparadores análogos, etc. Dentro de los más representativos están los A Ttiny 10, 11 y 12, microcontroladores FLASH de 1K de memoria de programa, 64 *bytes* de RAM y programables dentro del circuito.

Microcontroladores BASIC Stamp

La empresa *Parallax Inc.* produce estos microcontroladores que son muy potentes y poseen unas herramientas de programación que los hacen muy fáciles de manejar. Con ellos el diseñador no requiere de conocimientos avanzados en electrónica digital, ni en microprocesadores, ni en el desarrollo de programas en lenguaje ensamblador, entre otros. Por esta razón están siendo ampliamente utilizados por estudiantes desde el colegio hasta la universidad, por aficionados, técnicos e ingenieros de diferentes ramas y, además, se han popularizado mucho en el campo de la robótica experimental.

Los BASIC Stamp son pequeñas computadoras construidas en una tarjeta de circuito impreso, **figura 6.4** y su principal característica es que ejecutan programas escritos en un lenguaje muy similar al tradicional lenguaje BASIC, llamado PBASIC. Su forma de trabajo es muy simple: según el circuito que se quiera controlar, el diseñador escribe su programa en la computadora, lo compila y lo lleva hacia el microcontrolador a través de un cable de conexión que se ha preparado previamente. El microcontrolador recibe dichas instrucciones y las almacena en una memoria

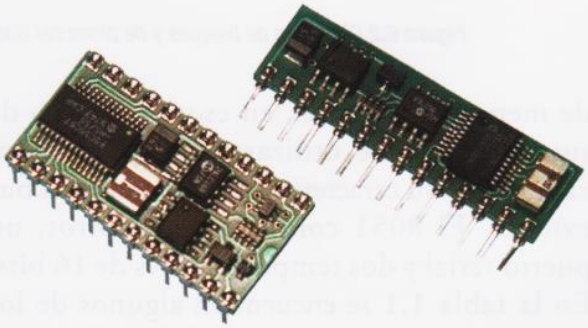


Figura 6.4 *Microcontroladores BASIC Stamp*

EEPROM, de donde las lee para ejecutarlas secuencialmente.

Los BASIC Stamp I y II incluyen un microcontrolador de la familia Microchip o Scenix, el cual está grabado previamente con un interpretador de códigos que le permite tomar los datos que se han pasado desde la computadora y ejecutar las órdenes correspondientes. También poseen una memoria EEPROM serial donde se almacena el programa recibido y que se debe ejecutar; un regulador de voltaje, su respectivo circuito oscilador y un sistema de reinicializar (*reset*) muy eficiente.

Los BASIC Stamp han evolucionado rápidamente desde su creación. El primer modelo disponible, conocido con el nombre de BASIC Stamp simplemente, posee un PIC16C56 como *chip* interpretador y una memoria EEPROM serial 93LC56 donde se almacena el programa. Este sistema, incluye un conector para batería de 9 voltios, donde se debe conectar la alimentación externa; también incluye un área de circuito impreso, libre para hacer prototipos y, por supuesto, los pines donde se conecta el cable que viene desde la computadora y a través del cual se descarga el programa que ha escrito el usuario. Este cable se conecta al puerto paralelo de la computadora. En la **figura 6.5** se muestra este modelo de BASIC Stamp con su respectivo cable de programación.

El BASIC Stamp I

Este microcontrolador presentó un avance sustancial respecto a su antecesor ya que está construido con tecnología de montaje superficial, lo que significa un ahorro de espacio muy importante en la aplicación final. En la **figura 6.6** se muestra su aspecto físico. Como se puede apreciar, posee un circuito integrado interpretador basado en un PIC16C56 y una memoria serial, un regulador de voltaje que permite alimentar el sistema con una fuente de 6 a 15 voltios y un sistema de *reset* que reinicializa el funcionamiento del microcontrolador en caso de que la fuente del sistema caiga por debajo de 4V.

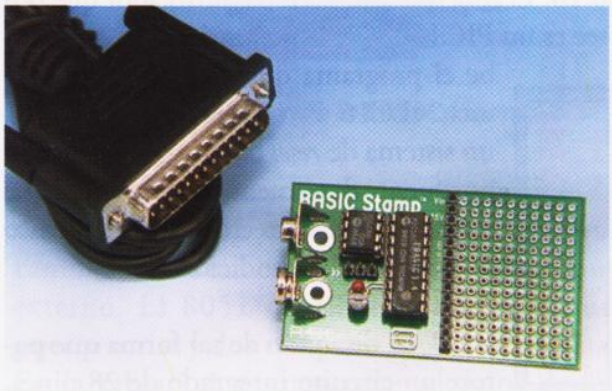


Figura 6.5 *Modelo original del BASIC Stamp con su cable de programación*

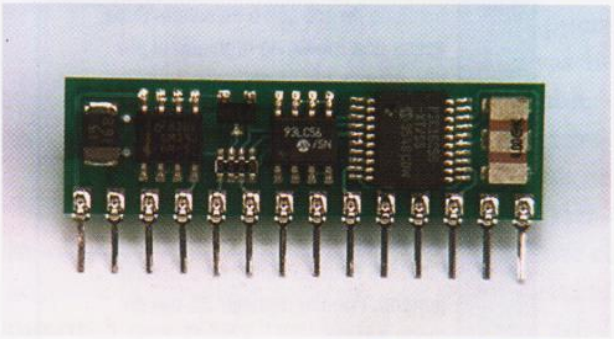
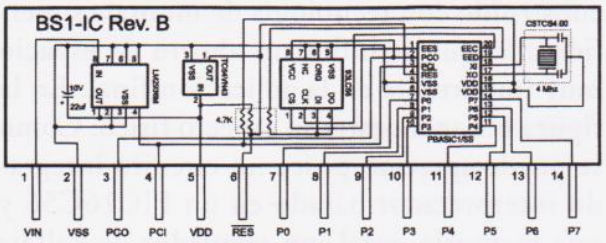


Figura 6.6 *BASIC Stamp I*

Como se puede apreciar en la **figura 6.7**, este sistema posee 14 pines organizados en una sola fila; de ellos, ocho se pueden utilizar como entradas o salidas individuales, posee un pin de *reset*, una entrada para alimentación con fuente regulada de 5V, una entrada para alimentación con fuente no regulada de 6 a 15 voltios, pin de tierra, y los pines de programación a través de los cuales se comunica con la computadora. En la **figura 6.8** se muestra la configuración del cable necesario para la programación; éste se debe conectar al puerto paralelo de la computadora.

Una alternativa muy práctica para montar los prototipos o circuitos de prueba con el BASIC Stamp I, es utilizar la tarjeta de prototipos (*carrier board*) diseñada por Parallax, en la cual se dispone de un conector para la fuente de alimentación, un circuito impreso para montar otros componentes y el conector para la respectiva programación. En la **figura 6.9** se muestra el diseño de dicha tarjeta.



VIN	Entrada de alimentación no regulada. Acepta de 6-15 VCC
GND	Tierra
PCO	Conectar al pin 11 del puerto paralelo para programación (BUSY)
PCI	Conectar al pin 2 del puerto paralelo para programación (DO)
VDD	5 voltios de entrada o de salida. Si se aplica una fuente no regulada al pin VIN, en este pin se obtienen 5 voltios. Si el pin VIN se deja al aire, se debe conectar aquí la fuente de 5V del sistema
RES	Reset de entrada/salida. Cae a un nivel bajo cuando la fuente de +5V está por debajo de 4V.
P0-P7	Pines de entrada salida de propósito general. Pueden manejar 25 mA en modo source.

Figura 6.7 *Distribución de pines del BASIC Stamp I*

En lo que respecta a la programación del microcontrolador, se deben tener en cuenta varios aspectos. El primero, tiene que ver con el programa (*software*) que se ejecuta en la computadora y que permite hacer todo el desarrollo de la aplicación. Este es un programa diseñado por Parallax y es muy similar a los editores de texto sencillos. En este sistema, se escriben las instrucciones para la aplicación particular y, mediante unos comandos definidos por combinaciones de teclas especiales, se puede compilar el programa, revisar la sintaxis y descargarlo hacia el microcontrolador.

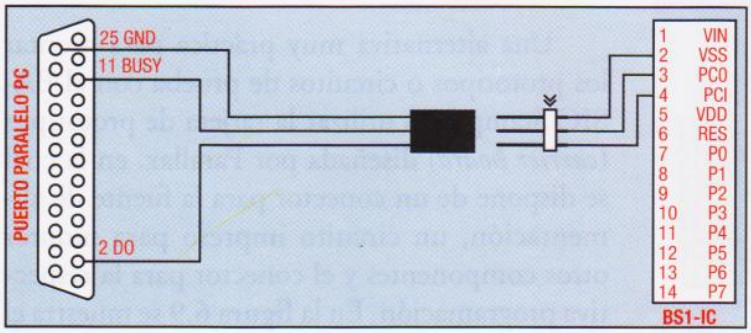


Figura 6.8 *Cable de programación del BASIC Stamp I*

Otro aspecto importante de la programación es el conocimiento que se debe tener sobre la capacidad de memoria del microcontrolador. Para el caso del BASIC Stamp I, se tienen 256 posiciones de memoria de programa y 16 *bytes* de memoria RAM; dentro de esta última está el control de los pines de entrada/salida y de las variables que se utilicen dentro del programa.

Como dijimos anteriormente, la característica más importante de los BASIC Stamp es la posibilidad de ejecutar programas en un lenguaje muy parecido al tradicional BASIC. Se trata de un lenguaje desarrollado por Parallax y que se denomina PBASIC. Con base en esto, el diseñador de aplicaciones puede utilizar muchas herramientas de programación que son difíciles de conseguir cuando se trabaja con microcontroladores corrientes, los cuales son programados en lenguaje ensamblador.

Dentro del grupo de instrucciones se encuentra la posibilidad de hacer ciclos condicionales, de generar sonidos, de hacer lecturas de botones o pulsadores, de medir tiempos o de generar pulsos con una sola instrucción, etc. Como se puede apreciar, son instrucciones muy poderosas y que disminuyen notablemente el tiempo de desarrollo de una aplicación.

El BASIC Stamp II

Éste posee el mayor número de características especiales. Su aspecto físico se puede ver en la **figura 6.10**. El *chip* interpretador o controlador que posee es un PIC16C57 y la memoria serial que recibe el programa que se va a ejecutar es una 24LC16. Este sistema posee además, un sistema de *reset* bastante sofisticado y una fuente de alimentación regulada que garantizan su correcto funcionamiento aún en difíciles condiciones.

Está diseñado de tal forma que parece un circuito integrado de 28 pines, de los cuales 16 son programables como

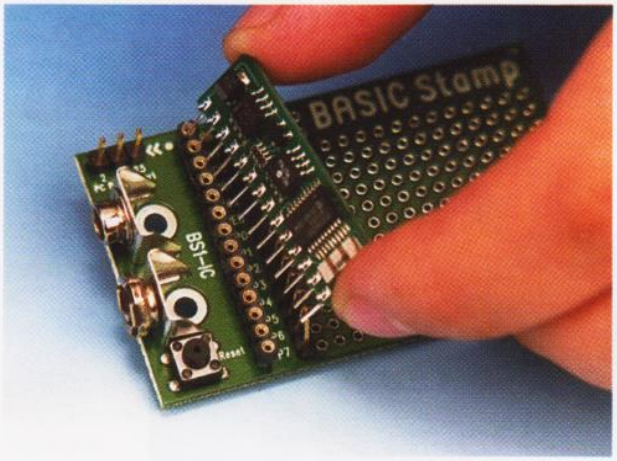


Figura 6.9 *Tarjeta para desarrollos con BASIC Stamp I*

entrada o salida; además, posee entrada para una fuente de alimentación regulada de 5V, una fuente de alimentación no regulada de 6 a 15 voltios, pin de *reset* y pines de comunicación serial que soportan niveles de voltaje RS-232. A través de estos últimos se hace la programación desde la computadora y se puede establecer comunicación con otros dispositivos externos.

Como se acaba de mencionar, la programación de este dispositivo se hace desde el puerto serial de la computadora, **figura 6.11**, a diferencia de los dos modelos anteriores que lo hacían desde el puerto paralelo. Al igual que el BASIC Stamp I, en este caso también se dispone de una tarjeta para el desarrollo de prototipos (*carrier board*), en la cual se pueden desarrollar aplicaciones particulares a la vez que

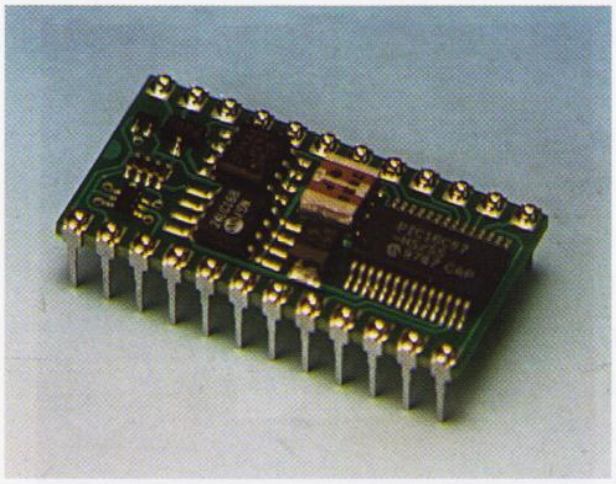


Figura 6.10 *BASIC Stamp II*

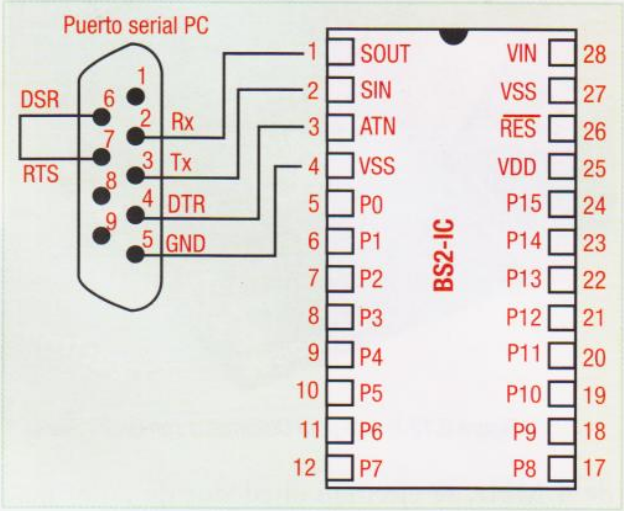


Figura 6.11 *Distribución de pines y cable de conexión del BASIC Stamp II*

se dispone de conexión para la fuente de alimentación y para el cable de programación. En la **figura 6.12** se puede ver la tarjeta de prototipos del BASIC Stamp II.

En lo que respecta a su programación, este dispositivo también posee su propio programa de desarrollo donde el diseñador crea sus programas, los compila y los descarga hacia el microcontrolador. Inclusive, existe un ambiente de desarrollo especial para trabajar bajo Windows que presenta las facilidades propias de este tipo de aplicaciones.

La memoria de este microcontrolador es mucho más amplia que la de sus predecesores; en la memoria de programa cuenta con 2048 posiciones de EEPROM y en la memoria RAM cuenta con 32 *bytes*. Sus instrucciones, aparte de las mismas del BASIC Stamp I, incluyen comandos especiales que le permiten manejar tonos DTMF, comunicaciones seriales RS-232 y PWM, entre otros.

Un aspecto muy importante que se debe tener presente a la hora de iniciar un diseño con los BASIC Stamp es que la frecuencia o reloj efectivo de operaciones es relativamente baja comparada con un microcontrolador normal. Es decir, si en un PIC con reloj de 4 MHz se ejecutan un millón de instrucciones cada segundo, en un BASIC Stamp, que posee el mismo cristal

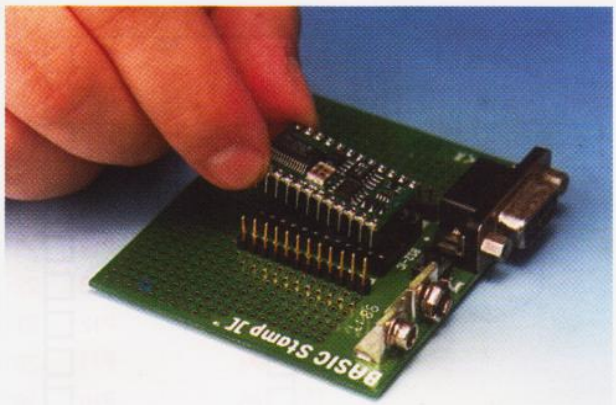


Figura 6.12 *Tarjeta para Desarrollos con BASIC Stamp II*

de 4 MHz, se ejecutan alrededor de 2000 instrucciones. Esto se debe a que cada instrucción del programa en PBASIC, se debe ejecutar en varias instrucciones de ensamblador del *chip* interpretador, que hace parte del BASIC Stamp.

Las versiones más actualizadas de los BASIC Stamp son el BS2p24-IC y el BS2p40-IC, que vienen en presentaciones de 24 y 40 pines, **figura 6.13**. Presentan toda una serie de nuevas funciones y comandos (55 distintos) y es muy interesante saber que las pistas de su circuito impreso están hechas en oro para mejorar su conductividad.

Las principales características las podemos apreciar en la **tabla 6.2**. Ahora poseen un microcontrolador SCENIX SX48AC a 20 MHz, en lugar del PIC y manejan fácilmente los protocolos I²C, DALLAS 1-WIRE y módulos LCD.

Hasta aquí hemos visto los aspectos generales de algunas de las principales familias de microcontroladores. Ahora estamos listos para iniciar un estudio detallado, siguiendo el orden de familias que se ha mostrado en las primeras lecciones, sobre la estructura básica, el funcionamiento y el manejo de cada una de las partes y módulos que hemos enunciado anteriormente y que forman parte de un microcontrolador. ¡Mucha dedicación y éxitos, son nuestros deseos para usted, que se inicia ahora de lleno en este maravilloso mundo de los microcontroladores!

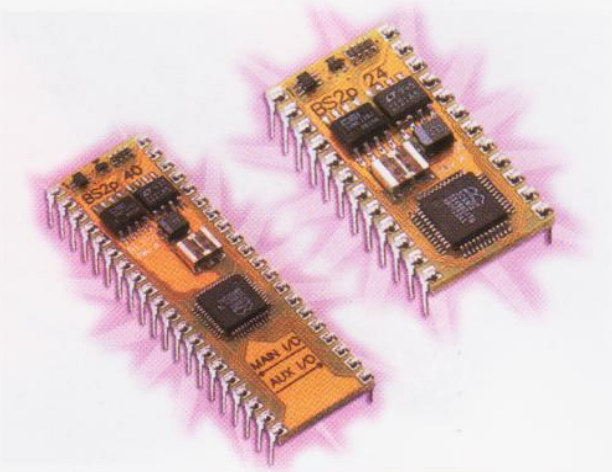


Figura 6.13 *Microcontroladores BASIC Stamp BS2p24 y BS2p40.*

Presentación	24 o 40 pines
Temperatura del ambiente	0 – 70 C
Microcontrolador	SCENIX SX48AC
Velocidad del procesador	20 MHz
RAM	38 bytes
EEPROM	8 x 2K bytes
Pines I/O	16 / 32
Consumo de corriente	40 mA en modo RUN y 400 uA en modo SLEEP

Tabla 6.2 *Características más representativas de los nuevos BASIC Stamp BS2p y BS2p40.*

CEKIT S.A. tiene disponibles una serie de entrenadores y tarjetas para prácticas con este tipo de microcontroladores, **figura 6.14.**

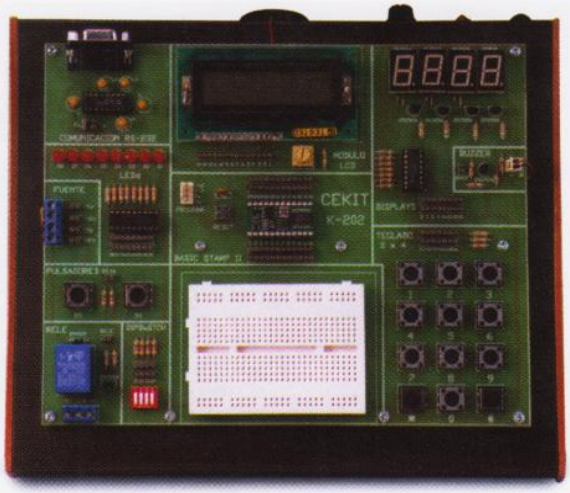
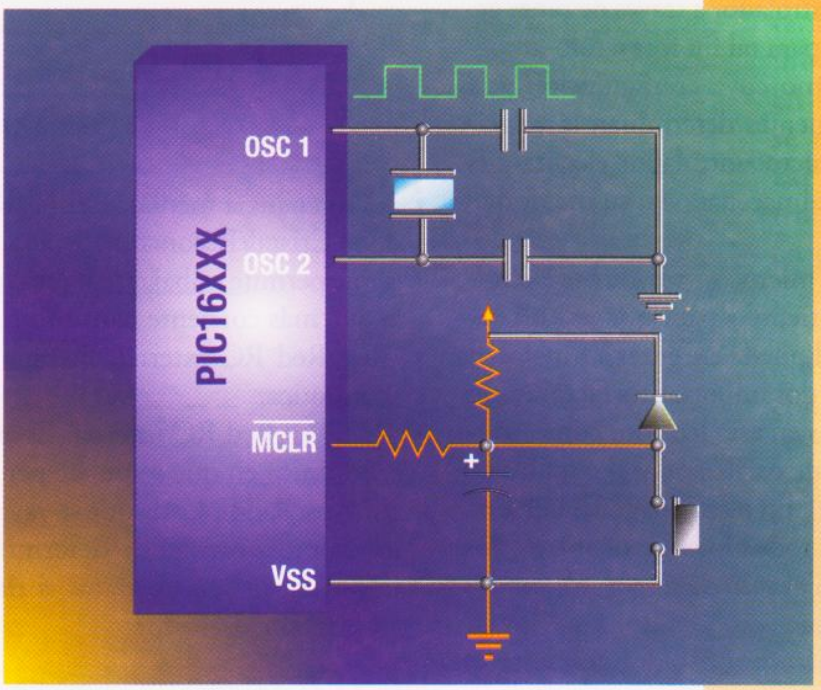


Tabla 6.14 *Entrenador para BASIC Stamp II, modelo K-202*



EL OSCILADOR Y EL CIRCUITO DE REINICIALIZACIÓN (RESET) EN LOS MICROCONTROLADORES PIC

Con esta lección iniciamos el estudio detallado de los principales componentes de cada una de las familias de los microcontroladores mencionados anteriormente en cuanto a su estructura eléctrica y funcionamiento. Empezamos con los circuitos de oscilación y de reinicialización (*reset*) de los PIC, dos elementos básicos para su funcionamiento cuyo conocimiento es esencial para el desarrollo de los circuitos de aplicación. También es muy importante conocer como cambian los parámetros de un sistema dependiendo de la configuración de estos dos circuitos.

Todos los microcontroladores requieren de un sistema que genere una señal de reloj que le permita a la CPU ejecutar secuencialmente las instrucciones. Ese sistema se puede conectar externamente al microcontrolador a través de pines específicos destinados para tal fin formando un pequeño circuito externo cuya configuración y el valor de sus componentes determinan la velocidad con la que el microcontrolador ejecutará el programa que tiene almacenado en su memoria.

Cada vez que el microcontrolador se energiza debe esperar un tiempo para que esa oscilación se estabilice y él pueda empezar a funcionar correctamente. Esos dos importantes factores son los que estudiaremos en esta lección: como configurar los circuitos de oscilación y el circuito de reinicialización (*reset*). En términos generales; esta teoría es válida para todos los microcontroladores de esta marca; en el caso de que algun modelo tenga requerimientos especiales al respecto, lo estaremos explicando en su debido momento.

El circuito oscilador

En el interior de los microcontroladores PIC existe un circuito de oscilación que permite generar la señal de reloj necesaria para que todo el dispositivo trabaje correctamente. Cada cuatro períodos de esa señal de reloj se genera un ciclo de instrucción, en otras palabras, se ejecuta una instrucción, **figura** 7.1. En estos microcontroladores existen varios tipos de configuración para el oscilador con el cual va a funcionar la aplicación, algunas más rápidas que otras; esa configuración se puede establecer dentro del programa o manualmente en el momento de programar el dispositivo.

Los tipos de oscilador más utilizados son:

- LP (baja potencia): cristal a baja frecuencia. Es la que menos corriente consume dentro de las tres opciones de trabajo con cristal.

Figura 7.1 *Ciclo de instrucción*

- XT (ganancia media): Cristal/Resonador.
- HS: Cristal/Resonador de alta frecuencia. Es la que permite trabajar más rápido y por lo tanto la que más corriente consume.
- RC: Red RC externa. Es una solución muy económica.
- INTRC: Red RC interna. Es la solución más económica en que se puede pensar para el circuito oscilador. Cada que se vaya a trabajar con un microcontrolador determinado, se debe mirar en su hoja de datos si tiene disponible este tipo de oscilador.

En el interior del microcontrolador existe un circuito que, dependiendo de la selección hecha entre LP, XT y HS afecta su ganancia interna y debido a eso se presentan los distintos rangos de frecuencia. Entre más alta sea la frecuencia de operación del microcontrolador, más alto será su consumo de potencia. Este es un concepto fundamental dentro del desarrollo de proyectos con microcontroladores, por lo tanto es aconsejable elegir el oscilador que proporcione la más baja frecuencia

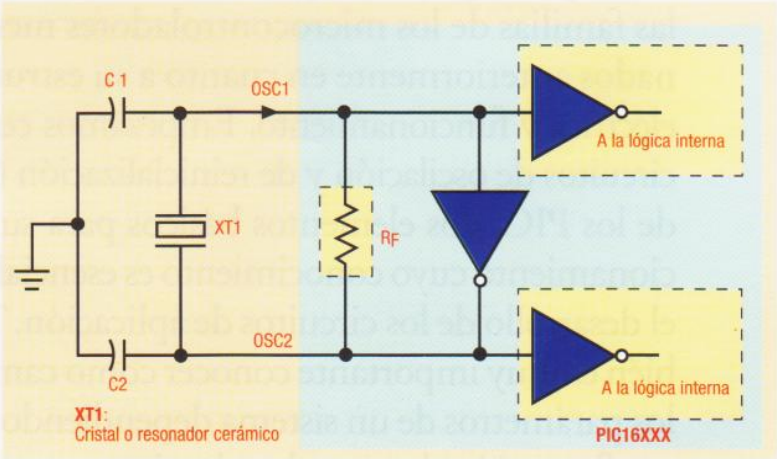


Figura 7.2 *Conexión del oscilador basado en cristal o resonador cerámico*

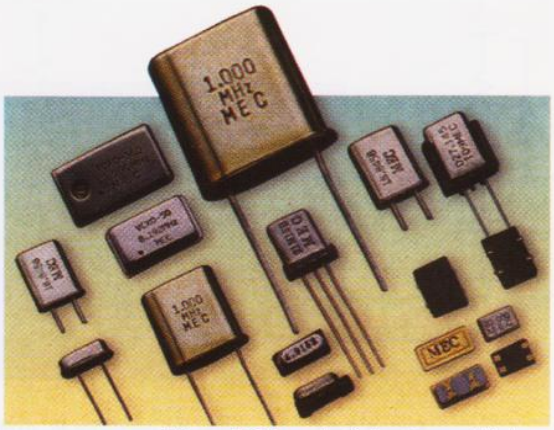


Figura 7.3 *Aspecto físico de un cristal común*

posible con el que nuestra aplicación y los dispositivos externos, puedan funcionar correctamente.

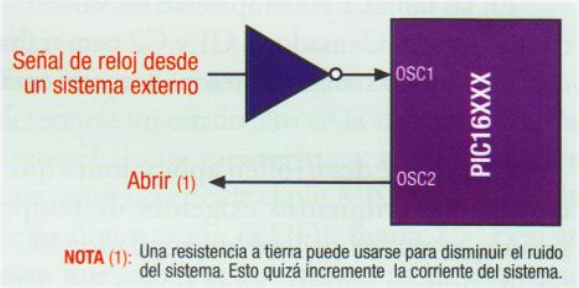
Osciladores basados en cristal o en resonadores cerámicos

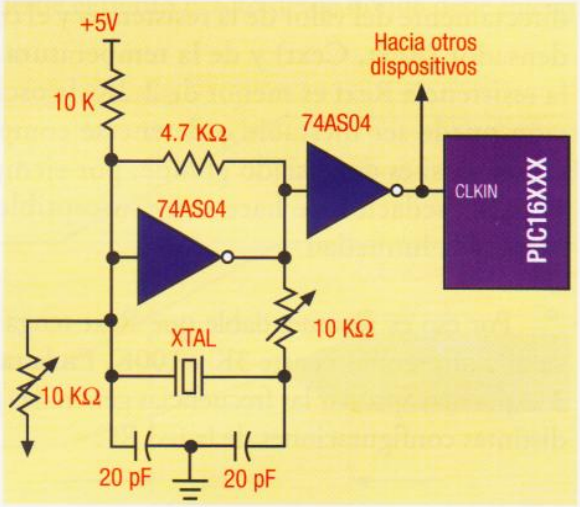
En este caso, para generar la señal de reloj, se debe conectar un pequeño circuito conformado por dos condensadores (C1 y C2) y un cristal o un resonador a los pines OSC1 y OSC2 del microcontrolador quienes se comunican internamente con un circuito lógico que genera la oscilación, **figura** 7.2.

Cuando se hace uso de cristales, estos se pueden conseguir en una gran variedad de encapsulados y frecuencias que por lo general vienen marcadas en su cápsula exterior, **figura** 7.3. Cuando se configura el modo de operación en XT, LP o HS, el microcontrolador puede tener una fuente externa de los pulsos de reloj conectada al pin OSC1. En la **figura** 7.4 se aprecian algunas posibles configuraciones de elementos externos que el lector puede ensayar para generar la señal de frecuencia externa. Estas configuraciones constan principalmente de compuertas lógicas, resistencias y condensadores.

Cuando el circuito de reloj recibe la alimentación, el oscilador empieza a trabajar, pero la realidad es que no lo hace de manera inmediata; requiere de cierto tiempo de estabilización. El tiempo requerido para que el oscilador opere normalmente depende de factores como la calidad del cristal, la temperatura del sistema, los valores de los condensadores utilizados, la frecuencia del cristal o resonador utilizado, la selección del modo de oscilador, el ruido del sistema, etc., **figura** 7.5.

La selección de los elementos que componen el circuito de oscilación no es al azar. Los valores de cada uno de esos elementos altera la salida final del circuito; por eso se debe hacer una cuidadosa selección de estos elementos teniendo en cuenta la velocidad de ejecución que requiera la aplicación que se esté desarrollando.





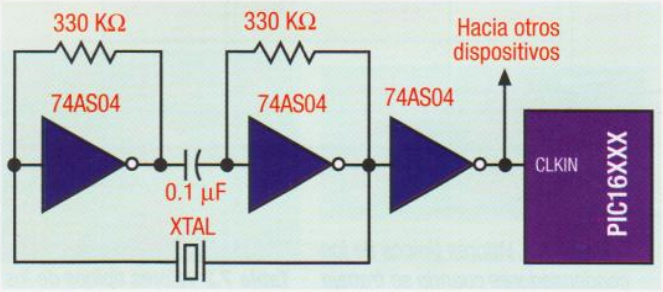


Figura 7.4 *Conexión de diversos osciladores externos*

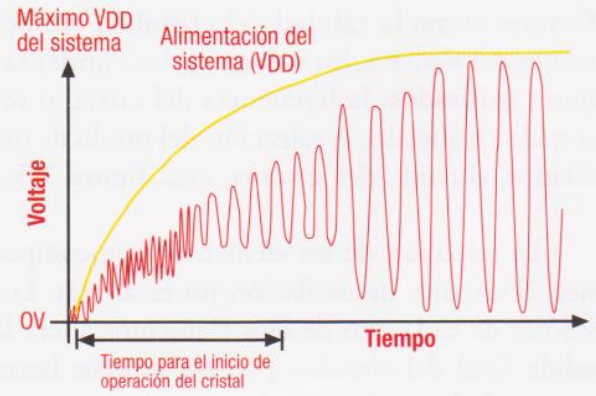


Figura 7.5 *Comportamiento del oscilador al recibir la alimentación e iniciar su operación*

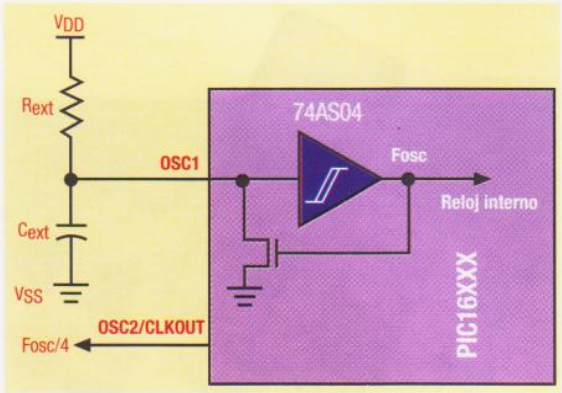


Figura 7.6 *Configuración del oscilador RC externo*

En las **tablas 1** y **2** se aprecian los valores típicos para los condensadores C1 y C2 para trabajar con un circuito basado en resonador y en cristal.

Cuando se desarrollen aplicaciones que no tengan requerimientos exigentes de temporización, se puede utilizar un oscilador RC externo como se muestra en la **figura** 7.6. En un oscilador de este tipo, la frecuencia depende directamente del valor de la resistencia y el condensador (Rext, Cext) y de la temperatura. Si la resistencia Rext es menor de 2.2K, la oscilación puede ser inestable o detenerse completamente; si es demasiado grande, por ejemplo 1M, la oscilación se hace muy susceptible al ruido y la humedad.

Por eso es recomendable que Rext tenga un valor comprendido entre 3K y 100K. En la **tabla 3** se pueden apreciar las frecuencias generadas por distintas configuraciones de la red RC.

En realidad este circuito puede funcionar inclusive sin condensador (Cext = 0) pero se recomienda utilizar valores por encima de 20pF por razones de estabilidad e inmunidad al ruido. Cuando el condensador es muy pequeño o no existe, la frecuencia de oscilación puede cambiar bruscamente debido a capacitancias creadas externamente como por ejemplo, la creada en el trazado del circuito impreso.

La frecuencia del oscilador, dividida por 4, se encuentra disponible como salida en el pin OSC2 y se puede utilizar para sincronizar otros dispositivos que trabajen en paralelo en el mismo circuito.

Circuito de reinicialización (reset)

En los microcontroladores PIC existe un pin denominado $\overline{\text{MCLR}}$. Cuando este pin se lleva a un nivel lógico bajo, el PIC entra al estado de reinicialización (*reset*). En este estado todas las salidas se apagarán y el reloj se desactivará. Todos los nuevos microcontroladores PIC poseen un filtro contra el ruido en el pin $\overline{\text{MCLR}}$ para detectar e ignorar pulsos pequeños, razón que facilita la incorporación de estos dispositivos en ambientes ruidosos o contaminados.

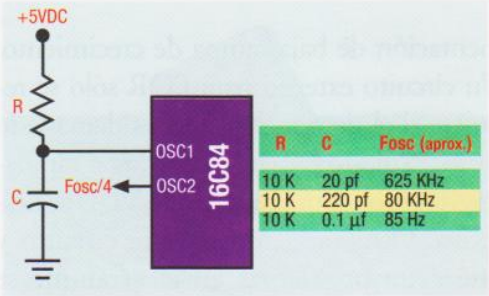
Dentro del mapa de memoria existe una dirección denominada **vector de** ***reset***, en esta di-

Modo	Frecuencia	C1/C2
XT	455 KHz 2.0 MHz 4.0 MHz	22 - 100 pF 15 - 68 pF 15 - 68 pF
HS	8.0 MHz 16.0 MHz	10 - 68 pF 10 - 22 pF

Tabla 7.1 *Valores típicos de los condensadores cuando se trabaja con resonadores*

Modo	Frecuencia	C1	C2
LP	32 KHz 200 KHz	68 - 100 pF 15 - 30 pF	68 - 100 pF 15 - 30 pF
XT	100 KHz 2 MHz 4 MHz	68 - 150 pF 15 - 30 pF 15 - 30 pF	150 - 200 pF 15 - 30 pF 15 - 30 pF
HS	8 MHz 10 MHz 20 MHz	15 - 30 pF 15 - 30 pF 15 - 30 pF	15 - 30 pF 15 - 30 pF 15 - 30 pF

Tabla 7.2 *Valores típicos de los condensadores cuando se trabaja con cristales*



R	C	Fosc (aprox.)
10 K	20 pf	625 KHz
10 K	220 pf	80 KHz
10 K	0.1 µf	85 Hz

Tabla 7.3 *Frecuencias de oscilación para distintos valores de la red RC*

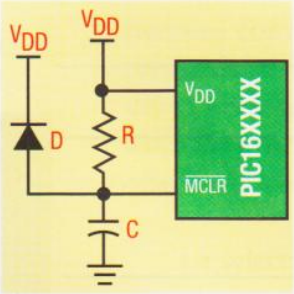


Figura 7.8a *Circuito externo de POR*

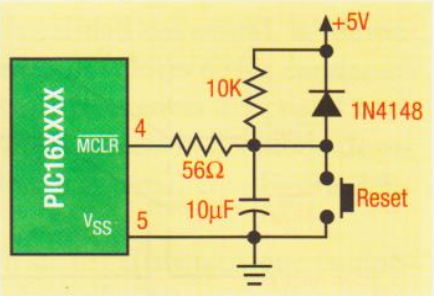


Figura 7.8b *Otro circuito para POR*

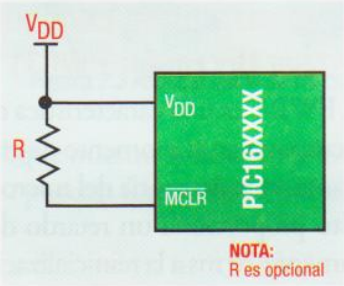


Figura 7.7 *Configuración para utilizar el POR*

rección es donde el PIC empieza la ejecución del programa. Esta dirección es la 0 para los PIC de medio y alto rango y corresponde a la última dirección para los PIC de rango bajo. Por ejemplo, el vector de *reset* de un PIC16C54 tiene la dirección 0x01FF. Cada vez que el PIC entra, por alguna razón, en estado de *reset*, la CPU del microcontrolador se dirige a esta posición de la memoria para iniciar nuevamente la ejecución de todas las instrucciones del programa.

Existen varias razones que pueden motivar un estado de *reset* además del estado lógico bajo en el pin $\overline{\text{MCLR}}$, ellas se conocen como tipos de reset, y son: la reinicialización al encendido (*Power On Reset*), y la reinicialización por caída en la alimentación (*Brown Out Reset*). También es importante analizar la secuencia de inicio que presentan los PIC cada que se da un estado de *reset* ya que allí existen algunos retardos que se han incorporado como características estándar en los microcontroladores PIC, por las ventajas que ofrecen. Analicemos cada uno de ellos.

Reinicialización al encendido (Power On Reset: POR)

Este estado de *reset* se origina en el momento de detectarse un estado alto en la alimentación. Para aprovechar esta característica, se puede conectar una resistencia desde el pin $\overline{\text{MCLR}}$ hasta el voltaje de alimentación (VDD), **figura** 7.7. Esto asegura que cada vez que el sistema es alimentado adecuadamente, el programa inicia su operación desde el punto exacto desde donde debe iniciar.

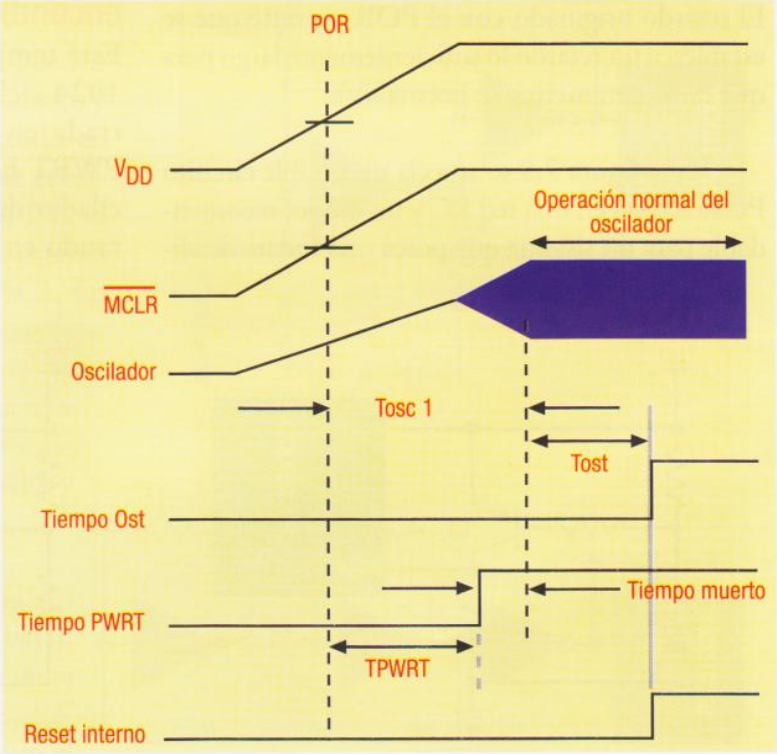


Figura 7.9 *Diagrama de tiempo del inicio de operación del oscilador*

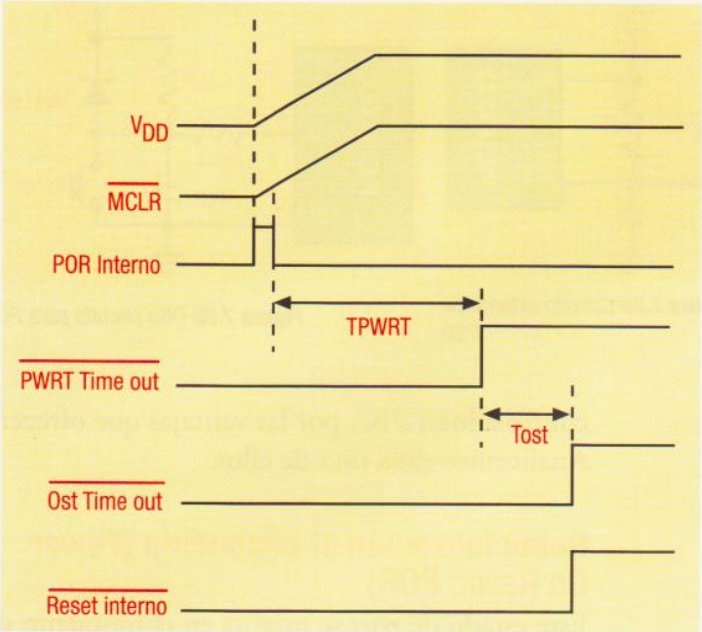


Figura 7.10 *Secuencia seguida al alimentar el sistema y el pin MLCR conectado directamente a VDD*

Cuando el dispositivo sale del estado de *reset* (comienza su operación normal), los parámetros de operación del dispositivo (voltaje, frecuencia, temperatura, etc.) deben encontrarse dentro de sus rangos normales de operación; de otra manera el microcontrolador no funcionará correctamente. El retardo originado con el POR permite que se establezca un retardo lo suficientemente largo para que estos parámetros se normalicen.

En la **figura** 7.8 se aprecia un posible circuito POR basado en una red RC y un diodo; recomendable para un sistema que posea una fuente de alimentación de baja rampa de crecimiento. Un circuito externo para POR sólo se requiere si el tiempo de VDD es demasiado lento. El diodo D ayuda a descargar rápidamente el condensador cuando VDD se desactiva. Otra variación de este circuito y que evita problemas en el arranque se muestra en la **figura** 7.8b.

Temporizador al encendido (Power Up Timer: PWRT)

El PWRT es una característica que se puede incorporar en el momento de grabar un programa en la memoria del microcontrolador. Esto proporciona un retardo de aproximadamente 70 ms a la reinicialización POR o a la BOR. El PWRT se basa en un oscilador RC interno dedicado. El microcontrolador permanecerá en estado de reset mientras el PWRT se encuentre activo. El retardo proporcionado por el PWRT le permite al voltaje de alimentación VDD alcanzar un nivel aceptable.

Temporizador de oscilación al encendido (OST)

Este temporizador proporciona un retardo de 1024 ciclos del oscilador (de acuerdo a la entrada en OSC1) después de que el retardo PWRT ha terminado. Esto asegura que el oscilador de cristal o resonador se encuentra operando en su modo estable.

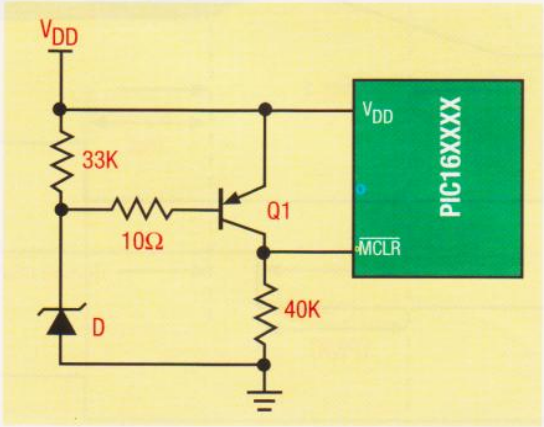


Figura 7.11 *Circuito 1 externo de Brown out*

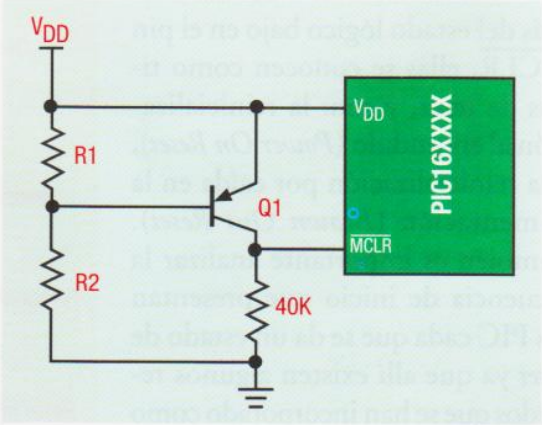


Figura 7.12 *Circuito 2 externo de Brown out*

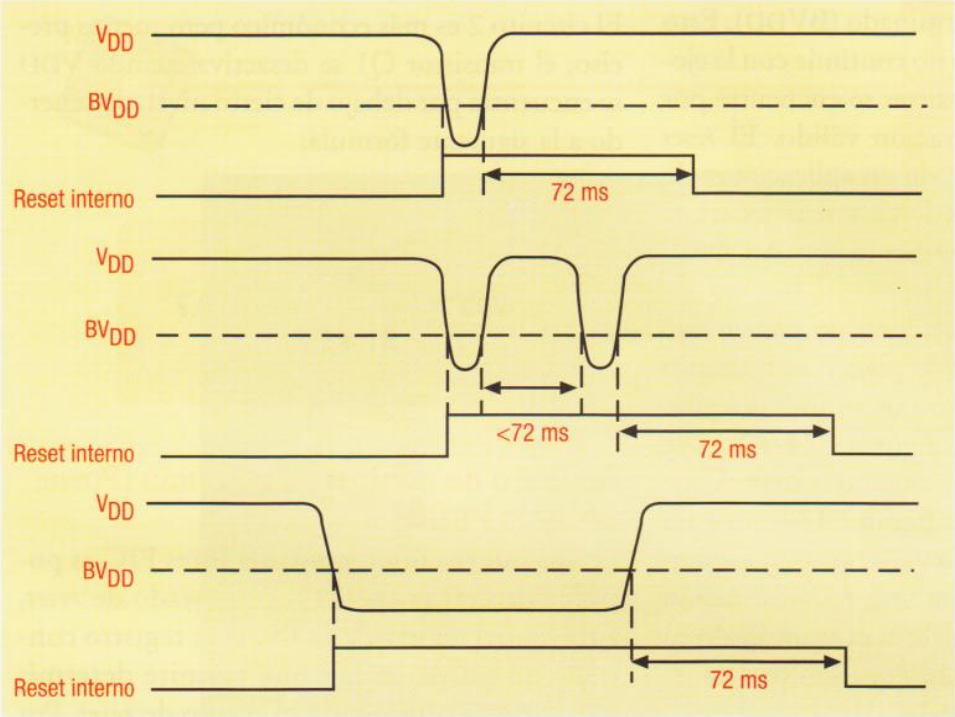


Figura 7.13 Comportamiento del BOR

En la **figura** 7.9 se observa la operación del circuito OST en conjunto con el PWRT. Allí podemos apreciar como a medida que el voltaje VDD crece también lo hace la señal en $\overline{MCLR}$; a partir de cierto nivel el POR empieza su operación y el PWRT se activa durante un tiempo; cuando éste termina, habilita al oscilador para que empieze su operación. Para cristales de baja frecuencia este tiempo de inicio puede ser un poco más largo.

Esto se debe a que el tiempo que toma el oscilador de baja frecuencia para empezar a oscilar es mayor que el retardo del PWRT. Así, el tiempo tomado desde que el PWRT se acaba hasta cuando el oscilador empieza a oscilar, se denomina tiempo muerto.

Secuencia seguida al alimentar el sistema

En el momento de alimentar el microcontrolador, se desarrolla la siguiente secuencia: primero se detecta el POR; si está habilitado, se invoca el PWRT. Después de que termina el tiempo del PWRT, se activa el OST, **figura** 7.10.

Cada vez que el pin $\overline{MCLR}$ tome un nivel alto, el proceso se repite automáticamente. Esa característica es muy útil, por ejemplo, en aplicaciones donde se necesiten sincronizar varios microcontroladores que trabajen en paralelo.

Si el voltaje final del sistema no cumple con las especificaciones eléctricas del microcontrolador (las cuales se pueden hallar en las hojas de datos del fabricante) el pin $\overline{MCLR}$ debe mantenerse en bajo hasta que el voltaje sea el adecuado. El uso de un retardo con una red RC externa debe ser suficiente para este propósito.

Reinicialización por caída en la alimentación (Brown-Out Reset: BOR)

Dentro de los microcontroladores PIC existe esteotro tipo de reset denominado ***Brown-out***, el cualpuede ser habilitado o deshabilitado por el usuario. Este permite llevar el PIC a un estado de *reset* cuando el voltaje de alimentación cae por

bit 6:3 No implantados
bit 1 **POR**: Bit de estado del reset Power-on
1 = No ha ocurrido reset por Power-on
0 = Ha ocurrido un reset Power-on
bit 0 **BOR**: Bit de estado del reset Brown-out
1 = No ha ocurrido un reset Brown-out
0 = Ha ocurrido un reset Brown-out

Figura 7.14 Registro PCON

debajo de un voltaje determinado (BVDD). Esto asegura que el dispositivo no continúe con la ejecución del programa mientras se encuentre por fuera del rango de operación válido. El *reset* **Brown-out** es muy utilizado en aplicaciones de líneas CA o en aplicaciones donde se conmuten cargas y donde el voltaje puede caer temporalmente.

Algunos microcontroladores no poseen este tipo de *reset,* para los que lo poseen es necesario ajustar el voltaje de **Brown-out** al nivel deseado. Para este propósito, en las **figuras** 7.11 y 7.12 se muestran dos ejemplos de circuitos externos que se pueden implantar. En la **figura** 7.13 se muestra un diagrama de tiempo donde se ilustran los momentos en que se origina una reinicialización BOR. Cada uno de ellos debe ser comprobado en la práctica para verificar que cumplan con los requerimientos de la aplicación.

En el caso del circuito 1 activará el *reset* cuando VDD se encuentre por debajo de VZ + 0.7. Donde Vz es el voltaje zener. Al utilizar este circuito la configuración ***Brown Out Reset*** debe deshabilitarse en el momento de la programación. Las resistencias se deben ajustar dependiendo de las características del transistor.

El circuito 2 es más económico pero menos preciso; el transistor Q1 se desactiva cuando VDD se encuentra por debajo de cierto nivel, de acuerdo a la siguiente fórmula:

$$V_{DD} * \frac{R1}{R1 + R2} = 0.7$$

Registro de control de potencia (Power CONtrol: PCON)

En los nuevos microcontroladores PIC es posible detectar la fuente de un estado de *reset*, a través del registro PCON. Este registro contiene un bit de estado que permite determinar que está originando el estado de *reset*. Por ejemplo, determina si fue un POR, un *reset* externo, etc. Este registro lo podemos observar en la **figura** 7.14, junto con los estados de los *bits* en el momento que se originan los distintos tipos de *reset*. En la **figura** 7.15 se muestra un diagrama de bloques genérico sobre como interactúan las diversos tipos de condiciones que originan un estado de *reset*.

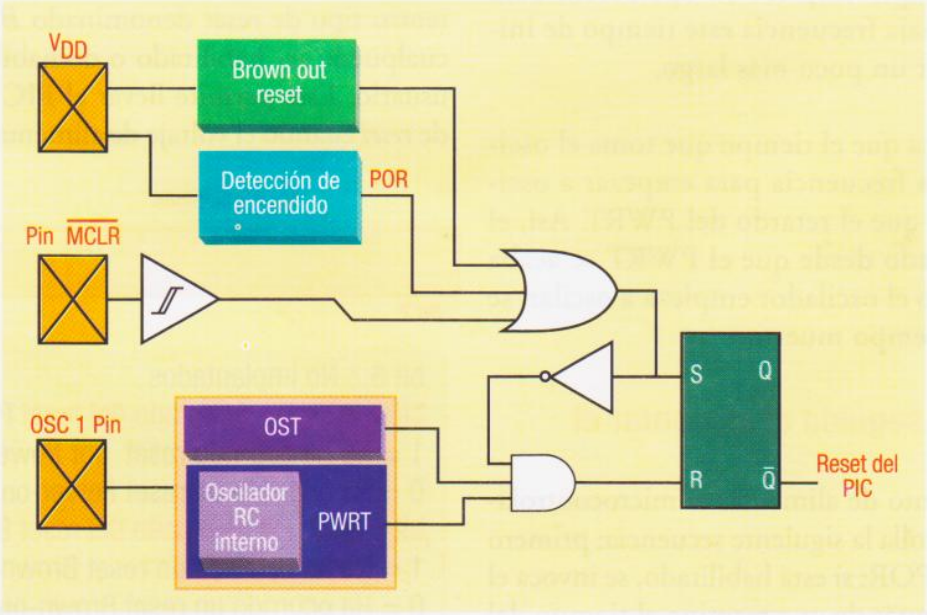
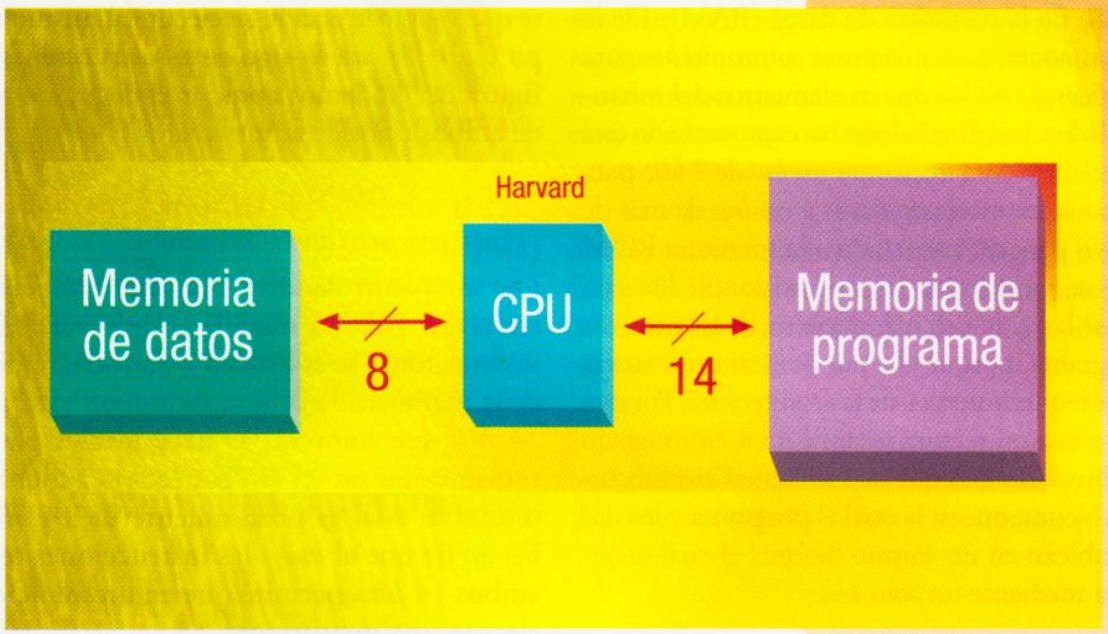


Figura 7.15 *Esquema general de los tipos de RESET en los microcontroladores PIC*

LECCIÓN 8



ARQUITECTURA DE LOS MICROCONTROLADORES PIC

En esta lección estudiaremos aspectos básicos de la estructura interna o arquitectura de los microcontroladores PIC, tales como la manera en que ejecutan las instrucciones teniendo en cuenta su arquitectura básica y el funcionamiento de la CPU y la ALU.

Los microcontroladores PIC poseen características que los hacen muy competitivos dentro del mercado global de la electrónica. Entre ellas se encuentran su arquitectura Harvard y su reducido conjunto de instrucciones (**RISC**: *Reduced Instruction Set Computer*), veamos los puntos más representativos de esta tecnología.

Arquitectura Harvard

Este tipo de arquitectura se caracteriza por tener la memoria de programa (ROM) en un bloque separado de la memoria de datos (RAM). De la misma manera, cada una posee su propio *bus* para comunicarse con los demás elementos del microcontrolador. Los diseñadores han aprovechado esta características para implantar un *bus* de 8 *bits* para conectar la memoria de datos y un *bus* de más de ocho *bits* para ser conectado a la memoria ROM donde se encuentran las instrucciones. De esta forma se hace un uso más eficiente de la memoria de programa, porque ella puede estar optimizada para los requerimientos de la arquitectura. Por esta razón, esta arquitectura permite un mejor manejo de las instrucciones que la tradicional arquitectura von Neumann, en la cual el programa y los datos se ubican en un mismo bloque, el cual se comunica mediante un solo *bus*.

Para ejecutar una instrucción, una máquina de arquitectura von Neumann debe realizar uno o más (normalmente más) accesos al bloque de memoria a través de un *bus* de 8 *bits*; la razón es que existen instrucciones cuya dimensión o longitud en *bits* es mayor de 8 *bits*, superando la longitud del *bus* de datos. Adicionalmente, los datos deben buscarse dentro de la RAM y llevar a cabo operaciones de distintos tipos, todo por el mismo *bus* de 8 *bits*. Así, al trabajar instrucciones y datos por un mismo *bus*, se originará demasiada congestión en éste.

Caso contrario ocurre en la arquitectura Harvard, en la cual una instrucción puede buscarse en un solo ciclo de instrucción, debido a que el *bus* que conecta la memoria de programa es de 14 *bits*. Al tiempo que la memoria de programa se está accediendo, la memoria de datos, como cuenta con su propio *bus*, puede ser leída o escrita. Esta configuración de *buses* separados permite que una instrucción se ejecute al mismo tiempo que se busca la siguiente instrucción. En la **figura 8.1** podemos apreciar el diagrama básico de ambos tipos de arquitecturas.

Instrucciones de palabra sencilla (*single word*)

Los microcotroladores PIC manejan instrucciones de palabra sencilla (*single word*). Estas instrucciones se codifican y almacenan dentro de la memoria de programa con un tamaño de 14 *bits*; ese número las hace ideales para ser transmitidas por el *bus* conectado a dicha memoria, el cual es precisamente de 14 *bits*. El hecho de que el *bus* y las instrucciones tengan ambos 14 *bits*, permite que en un solo ciclo se ejecute una instrucción.

Caso contrario ocurre en la arquitectura von Neumann; en ella las instrucciones por lo general son de múltiples *bytes* por lo que no es seguro que en cada posición de memoria exista una instrucción válida.

Manejo de las instrucciones (*pipeline*)

Para agilizar el proceso de ejecución de las instrucciones en los sistemas digitales, se ha diseñado un sistema en el cual se empieza la búsqueda de una instrucción a medida que se va completando la ejecución de la instrucción precedente o anterior. Este tipo de conexión o secuencia se conoce con el nombre de *pipeline* (entubar, encauzar, dirigir o distribuir por tubos).

Figura 8.1 *Diagrama de bloques de las arquitecturas Harvard y von Neumann*

En los microcontroladores PIC se utiliza esta técnica y los procesos de búsqueda y ejecución se traslapan. La búsqueda de una instrucción toma un ciclo de máquina y la ejecución se toma otro ciclo. Pero, en lugar de buscar la instrucción actual y ejecutar la instrucción previa, en cada ciclo de máquina se busca una instrucción al tiempo que se ejecuta otra.

Debido a que el *bus* de la memoria de programa tiene un tamaño de 14 *bits*, una instrucción se toma en un solo ciclo de máquina, y se ejecuta en otro ciclo. Si el resultado de la instrucción modifica el contenido del contador de programa, puede existir un pequeño retardo de un ciclo, el cual debe corregirse dentro de la secuencia, este caso lo estudiaremos más adelante.

Conjunto reducido de instrucciones

Los microcontroladores PIC se clasifican dentro de la categoría RISC (*Reduced Instruction Set Computer*); esto quiere decir que se requieren pocas instrucciones para llevar a cabo cualquier tarea. Es de resaltar aquí que un conjunto reducido de instrucciones puede ser aprendido más rápido por el usuario.

Estructura interna dividida en registros

La memoria de datos de los microcontroladores PIC se divide en registros, cada uno con una ubicación determinada, lo que permite que pueda accederse a cualquier parte de ella a través de direcciones. Existen algunos registros que cumplen funciones especiales los cuales también se encuentran ubicados dentro de ella en posiciones determinadas, como veremos en la siguiente lección denominada "La memoria en los microcontroladores PIC".

Instrucciones ortogonales (simétricas)

Las instrucciones de los microcontroladores PIC han sido diseñadas para que cumplan con cierta ortogonalidad o simetría; esto quiere decir que para cada instrucción existe otra instrucción simétrica que cumple la labor complementaria; esta característica también ayuda a facilitar el aprendizaje de las instrucciones.

Las instrucciones ortogonales hacen posible llevar a cabo cualquier operación sobre cualquier registro usando algún tipo de direccionamiento. La naturaleza simétrica y la ausencia de instruc-

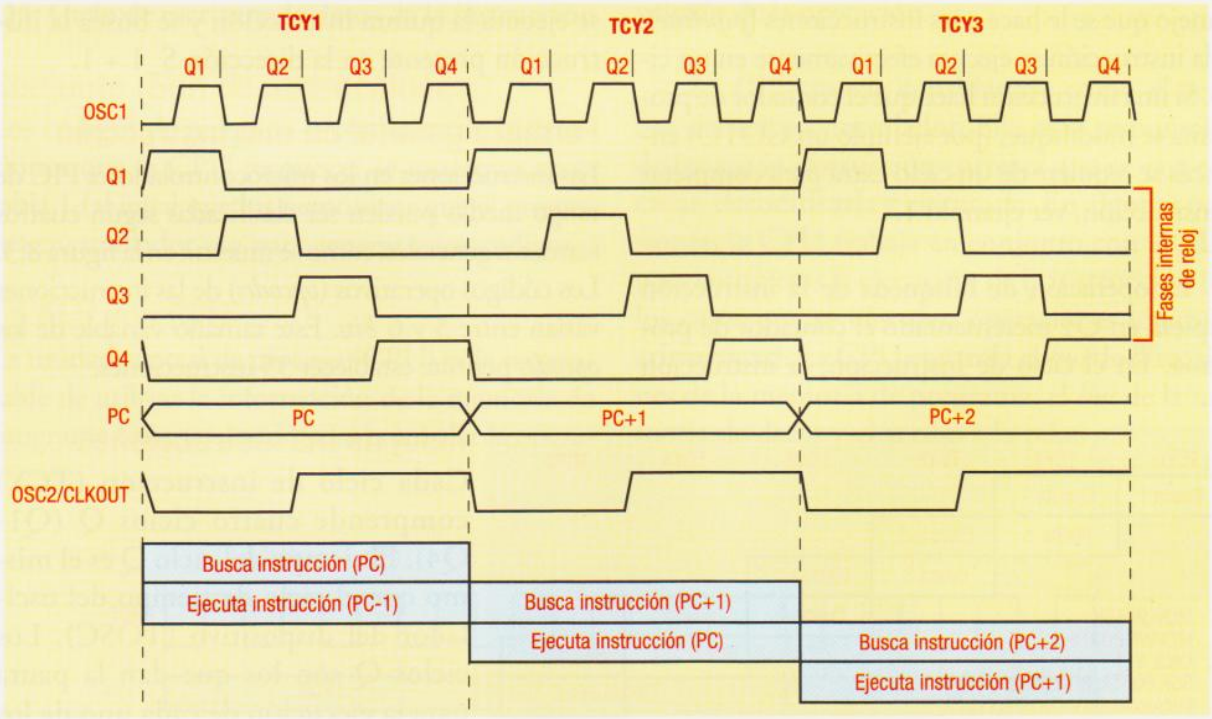


Figura 8.2 *Flujo de ejecución de una instrucción*

ciones especiales, (como ocurre en otros microcontroladores tipo CISC, quienes cuentan con numerosas instrucciones, donde algunas de ellas cumplen con labores muy puntuales o especiales), hacen que la programación sea muy simple y eficiente, a la vez que la curva de aprendizaje se reduce significativamente.

El ciclo de instrucción

La señal de entrada de reloj (pin OSC1) se divide internamente por cuatro, generando cuatro pulsos periódicos en cuadratura (desfasados) denominados Q1, Q2, Q3 y Q4. Internamente el contador de programa (PC) se incrementa cada vez que aparece Q1, la instrucción se busca en la memoria de programa y se almacena dentro del registro de instrucciones en Q4. Luego la instrucción se decodifica y se ejecuta durante el siguiente ciclo de Q1 a Q4. En la **figura 8.2**, se ilustra el flujo de la ejecución de una instrucción.

Un **ciclo de instrucción** está formado por cuatro ciclos Q (Q1, Q2, Q3 y Q4). La operación de búsqueda requiere de un ciclo de instrucción mientras hace la decodificación y la operación de ejecución toma otro ciclo de instrucción. Debido al manejo que se le hace a las instrucciones (*pipeline*), cada instrucción se ejecuta efectivamente en un ciclo. Si una instrucción hace que el contador de programa se modifique, (por ejemplo un GOTO) entonces se requiere de un ciclo extra para completar la instrucción, ver ejemplo 1.

La operación de búsqueda de la instrucción empieza en Q1 incrementado el contador de programa. En el ciclo de instrucción, la instrucción buscada es almacenada dentro de un registro de instrucción (IR) en el ciclo Q1. En ese momento ella se decodifica y se ejecuta durante los ciclos Q2, Q3 y Q4. La memoria de datos se ha leído durante el ciclo Q2 y se ha escrito durante el ciclo Q4.

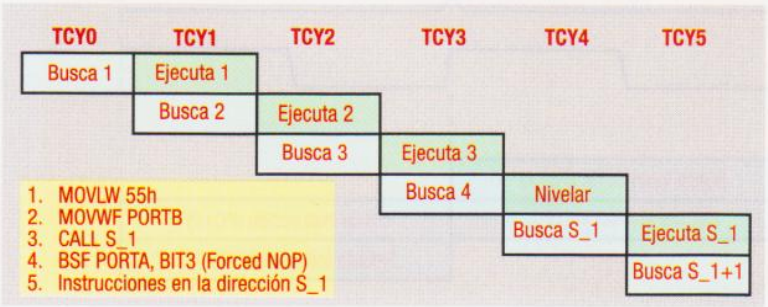
En el **ejemplo 1** se muestra la secuencia de operación (*pipeline*) de una instrucción. En TCY0 se busca la primera instrucción en la memoria de programa; en TCY1 ésta se ejecuta mientras se busca la segunda. Durante TCY2, la segunda instrucción se ejecuta mientras se busca la tercera. Durante TCY3 se busca la cuarta instrucción mientras se ejecuta la tercera (CALL S_1).

Cuando se ha terminado la ejecución de la tercera instrucción, la CPU hace que la dirección de la cuarta instrucción se lleve a la pila cambiando el contador de programa (PC) a la dirección de S_1. Esto significa que la instrucción que se buscó durante TCY3, debe ser nivelada desde el *pipeline*.

Durante TCY4 se nivela la cuarta instrucción (se ejecuta como un NOP: instrucción de no operación) y se busca la instrucción ubicada en la dirección S_1. Finalmente, durante TCY5, se ejecuta la quinta instrucción y se busca la instrucción presente en la dirección S_1 + 1.

Formato general de una instrucción

Las instrucciones en los microcontroladores PIC de rango medio pueden ser clasificadas según cuatros formatos generales, como se muestra en la **figura 8.3**. Los códigos operativos (*opcodes*) de las instrucciones varían entre 3 y 6 *bits*. Este tamaño variable de los *opcodes* permite establecer 35 instrucciones.



Ejemplo 1 *Secuencia de búsqueda y ejecución de las instrucciones*

Reloj de las instrucciones

Cada ciclo de instrucción (TCY) comprende cuatro ciclos Q (Q1-Q4). El tiempo del ciclo Q es el mismo que el ciclo de tiempo del oscilador del dispositivo (TOSC). Los ciclos Q son los que dan la pauta para la ejecución de cada uno de los procesos de decodificación, lectura,

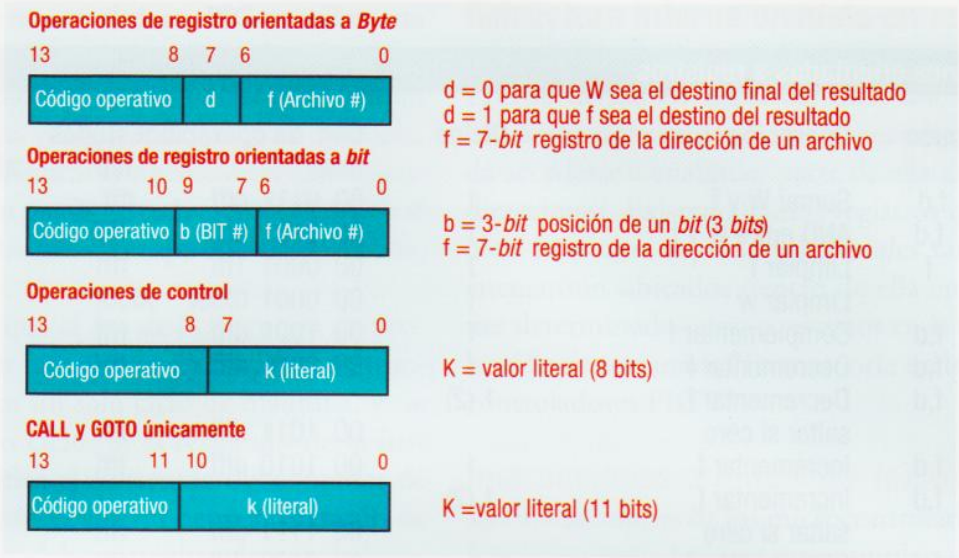


Figura 8.3 *Formato general de una instrucción*

procesamiento de datos, escritura, etc., de cada ciclo de instrucción, figura 8.4.

Los cuatro ciclos Q que conforman un ciclo de instrucción (TCY) se pueden generalizar así:

Q1: Ciclo de decodificación de la instrucción
Q2: Ciclo de lectura de datos de la instrucción
Q3: Procesamiento de datos
Q4: Ciclo de escritura de datos de la instrucción

Conjunto (*Set*) de instrucciones

Los códigos de máquina o conjunto de instrucciones que la CPU reconoce, se muestran en la tabla 1 (al igual que los nemónicos que el programa ensamblador usa para generar estos códigos).

La ALU y la CPU

La unidad central de proceso (CPU) es la responsable de utilizar la información de la memoria de programa (instrucciones) para controlar la operación del dispositivo. Muchas de estas instrucciones operan sobre la memoria de datos para lo cual se requiere de la unidad aritmético lógica (ALU, *Arithmetic Logical Unit*). Para llevar a cabo las operaciones aritméticas y lógicas, la ALU controla los *bits* de estado (los cuales se localizan en un registro especial denominado registro de estado). El resultado de algunas instrucciones obligan a los *bits* de estado a cambiar su valor, dependiendo del resultado de la operación.

La CPU puede considerarse como el cerebro del microcontrolador. Ella es la responsable de buscar la instrucción correcta que se va a ejecutar, decodificarla y ejecutarla. En algunas ocasiones, la CPU trabaja en conjunto con la ALU para completar la ejecución de la instrucción (en los casos que se efectúen operaciones lógicas o aritméticas). La CPU controla el *bus* de direcciones de la memoria de programa, el *bus* de la memoria de datos y el acceso a la pila.

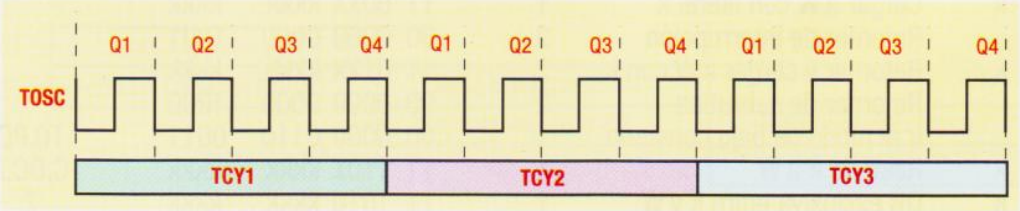


Figura 8.4 *Ciclos Q*

Nemotécnico		Operación	Ciclos	Cód. de operación msb	lsb	Estados afectados
Operaciones orientadas a registros						
ADDWF	f,d	Sumar W y f	1	00 0111 dfff	ffff	C,DC,Z
ANDWF	f,d	AND entre W y f	1	00 0101 dfff	ffff	Z
CLRF	f	Limpiar f	1	00 0001 1fff	ffff	Z
CLRW		Limpiar w	1	00 0001 0XXX	XXXX	Z
COMF	f,d	Complementar f	1	00 1001 dfff	ffff	Z
DECF	f,d	Decrementar f	1	00 0011 dfff	ffff	Z
DECFSZ	f,d	Decrementar f, saltar si cero	1 (2)	00 1011 dfff	ffff	
INCF	f,d	Incrementar f	1	00 1010 dfff	ffff	Z
INCFSZ	f,d	Incrementar f, saltar si cero	1 (2)	00 1111 dfff	ffff	
IORWF	f,d	OR entre W y f	1	00 0100 dfff	ffff	Z
MOVF	f,d	Mover f	1	00 1000 dfff	ffff	Z
MOVWF	f	Mover W a f	1	00 0000 1fff	ffff	
NOP		No operación	1	00 0000 0XX0	0000	
RLF	f,d	Rotar a la izquierda a través del carry	1	00 1101 dfff	ffff	C
RRF	f,d	Rotar a la derecha a través del carry	1	00 1100 dfff	ffff	C
SUBWF	f,d	Restar W de f	1	00 0010 dfff	ffff	C,DC,Z
SWAPF	f,d	Intercambiar nibbles de f	1	00 1110 dfff	ffff	
XORWF	f,d	OR exclusiva entre W y f	1	00 0110 dfff	ffff	Z
Operaciones orientadas a bits						
BCF	f,b	Limpiar bit b de f	1	01 00bb bfff	ffff	
BSF	f,b	Activar bit b de f	1	01 01bb bfff	ffff	
BTFSC	f,b	Probar bit b de f, saltar si es cero	1 (2)	01 10bb bfff	ffff	
BTFSS	f,b	Probar bit b de f, saltar si es uno	1 (2)	01 11bb bfff	ffff	
Operaciones literales y de control						
ADDLW	k	Sumar literal k a W	1	11 111X kkkk	kkkk	C,DC,Z
ANDLW	k	AND entre k y W	1	11 1001 kkkk	kkkk	Z
CALL	k	Llamar subrutina	2	10 0kkk kkkk	kkkk	
CLRWDT		Limpiar WDT	1	00 0000 0110	0100	TO,PD
GOTO	k	Salta a dirección k	2	10 1kkk kkkk	kkkk	
IORLW	k	OR entre k y W	1	11 1000 kkkk	kkkk	Z
MOVLW	k	Cargar a W con literal k	1	11 00XX kkkk	kkkk	
RETFIE		Retornar de interrupción	2	00 0000 0000	1001	
RETLW	k	Retornar y cargar a W con k	2	11 01XX kkkk	kkkk	
RETURN		Retornar de subrutina	2	00 0000 0000	1000	
SLEEP		Ir al modo de bajo consumo	1	00 0000 0110	0011	TO,PD
SUBLW	k	Restarle k a W	1	11 110X kkkk	kkkk	C,DC,Z
XORLW	k	OR exclusiva entre k y W	1	11 1010 kkkk	kkkk	Z

Tabla 1 *Conjunto de instrucciones de los microcontroladores PIC de rango medio*

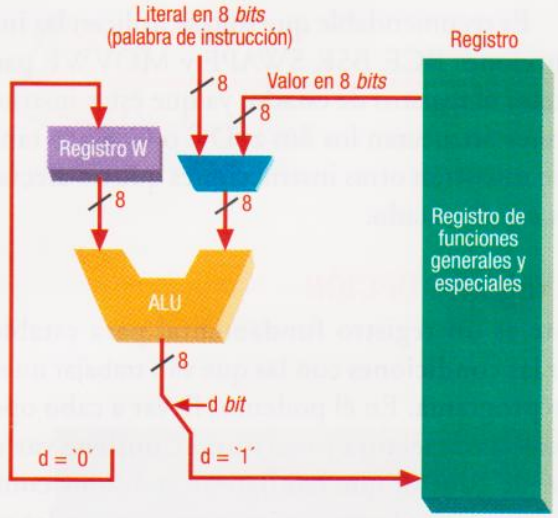


Figura 8.5 *Operación de la ALU y el registro W*

Unidad aritmética lógica (ALU)

Los microcontroladores PIC contienen una ALU y un registro de trabajo de 8 *bits*. La ALU es la unidad lógica y aritmética la cual desarrolla operaciones aritméticas y funciones booleanas entre los datos del registro de trabajo y cualquier registro.

La ALU es capaz de hacer sumas, restas, desplazamientos y operaciones lógicas. Las operaciones aritméticas utilizan el método numérico del complemento a dos. En instrucciones que manejan dos operandos, normalmente un operando es el registro de trabajo (denominado W) y el otro operando es un registro o una constante, **figura 8.5**.

El registro W es un registro no direccionable de 8 *bits* utilizado para las operaciones de la ALU. Dependiendo de la instrucción que se va a ejecutar, la ALU puede afectar los valores de los *bits carry* (C), *digit carry* (DC), y *zero* (Z) del registro de estado. Los *bits* C y DC, indican el momento en que en una operación de substracción se da un caso de "llevo" o "tomo prestado".

Los datos de la memoria RAM interactúan con las palabras de instrucción al llegar a la ALU. Dependiendo del valor de d en la correspondiente instrucción, el resultado final se dirige a w o al registro de la RAM, f.

El registro de estado

El registro de estado, **figura 8.6**, contiene el estado aritmético de la ALU, el estado del proceso

Registro: STATUS

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

bit 7 — bit 0

Dirección: 03h
Condición de *reset*: 000??XXX

IRP: Selector de página para direccionamiento indirecto
Este *bit* no se utiliza efectivamente en el PIC16F84, por lo que se puede utilizar como un *bit* de propósito general.

RP1,0: Selectores de página para direccionamiento directo.
Solamente RP0 se utiliza en el PIC16F84. RP1 se puede utilizar como un *bit* de propósito general.

T0: *Time Out* o *Bit* de finalización del temporizador.
Se coloca en 0 cuando el circuito de vigilancia *Watchdog* finaliza la temporización.

PD: *Power Down* o *Bit* de bajo consumo. Se coloca en 0 por la instrucción *SLEEP*.

Z: *Zero* o *Bit* de cero. Se coloca en 1 cuando el resultado de una operación lógica o aritmética es cero.

DC: *Digit Carry* o *Bit* de acarreo de dígito. En operaciones aritméticas se activa cuando hay acarreo entre el *bit* 3 y el 4.

C: *Carry* o *Bit* de acarrreo. En instrucciones aritméticas se activa cuando se presenta acarreo desde el *bit* más significativo del resultado.

Figura 8.6 *Registro de estado*

de reinicialización y los *bits* de selección de los bancos de memoria. Este concepto se estudiará en la siguiente lección: La memoria en los microcontroladores PIC.

El registro de estado puede servir para almacenar los datos de una instrucción, es decir, puede trabajar como cualquier otro registro. Si el registro de estado sirve de destino para una instrucción que afecte los *bits* C, DC o Z, entonces se deshabilita la escritura en estos tres *bits*. Esos son puestos en uno o en cero de acuerdo a la lógica del dispositivo y a las operaciones que ejecute. De hecho, los *bits* TO y PD no son escribibles.

El resultado de una instrucción que tenga como destino el registro de estado, puede arrojar un resultado distinto al esperado. Por ejemplo, si se ejecuta la instrucción CLRF ESTADO se borrarán los tres *bits* más significativos y Z se pondrá en uno. Esto deja al registro de estado con **000n n1nnn** (donde n significa: no cambia).

Es recomendable que solo se utilicen las instrucciones BCF, BSF, SWAPF y MOVWF para alterar el registro de estado, ya que estas instrucciones no afectan los *bits* z, DC o C. En la tabla 1 se muestran otras instrucciones que no afectan los *bits* de estado.

El registro OPCIÓN

Este es un registro fundamental para establecer las condiciones con las que va a trabajar nuestro programa. En él podemos llevar a cabo operaciones de lectura y escritura. Contiene varios *bits* de control que configuran módulos como los temporizadores, las interrupciones y las resistencias de *pull-up* internas del microcontrolador, **figura 8.7**.

Los *bits* PSA, PS0, PS1, PS2, INTDEG y RTS configuran el funcionamiento del temporizador interno (esto lo estudiaremos más adelante), el *bit* RBPU permite que habilitemos una resistencias de *pull-up* internas conectadas al puerto B.

Registro: OPTION (OPCIÓN)

$\overline{\text{RBPU}}$	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0
bit 7							bit 0

Dirección: 81h
Condición de *reset*: 11111111b

RBPU: *PortB Pull-up Enable* o Habilitación de *pull-up* del puerto B.
0: habilita las *pull-ups* internas
1: las deshabilita

INTEDG: *INT Interrupt Edge Select* o Selector de flanco de la interrupción INT
0: flanco de bajada
1: flanco de subida

RTS: *TMR0 Signal Source* o Fuente de señal de TMR0.
0: ciclo de instrucciones interno (Temporizador)
1: transición en el pin RA4/TOCK (Contador)

RTE: *TMR0 Signal Edge* o Flanco de la señal TMR0
0: incremento en transición de bajo a alto
1: incremento en transición de alto a bajo

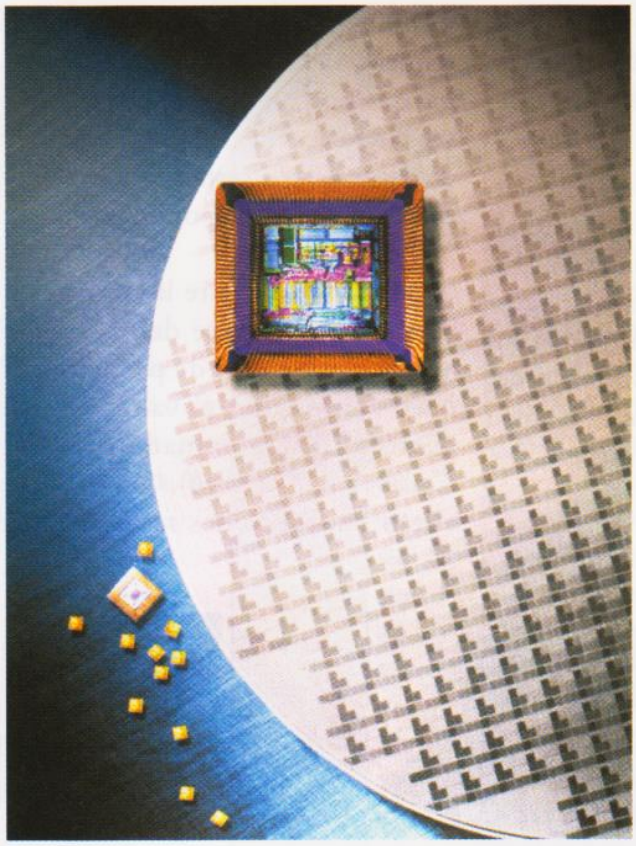
PSA: *Prescaler Assignment* o Asignación del preescalador
0: TRM0 (Contador/Temporizador)
1: WDT (Circuito de vigilancia)

PS2,1,0: *Prescaler Value* o Valores del preescalador.

Valor	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Figura 8.7 *Registro OPCIÓN*

LECCIÓN 9



LA MEMORIA EN LOS MICROCONTROLADORES PIC

Conocer cómo está estructurada la memoria dentro de los microcontroladores es un paso fundamental para desarrollar aplicaciones óptimas y aprovechar al máximo lo que estos circuitos integrados nos pueden ofrecer. En esta lección analizaremos la disposición y manejo de las memorias RAM y ROM internas de los microcontroladores PIC.

Podemos reseñar dos bloques principales en los que se divide la memoria en los microcontroladores PIC: la memoria de programa (ROM) y la memoria de datos (RAM). También existe una memoria muy importante denominada la **EPROM de datos**; ella será tema de estudio en la siguiente lección. Como vimos en la lección anterior, cada bloque cuenta con su propio *bus* de acceso, de manera que ambos pueden consultarse durante un mismo ciclo del reloj.

La memoria de datos, a su vez se divide en dos bloques: los registros de propósito general y los registros de funciones especiales (SFR: *Special Function Registers*). Los registros de propósito general son los que el usuario tiene a su disposición para ubicar las distintas variables de su programa, mientras que los registros SFR controlan muchas funciones del microcontrolador y de sus módulos periféricos.

El contador de programa

El contador de programa (PC) es un registro de 13 *bits* que especifica la dirección en la memoria de programa de la instrucción que debe buscarse para ser ejecutada. El *byte* bajo del PC se denomina registro PCL y sobre él se pueden realizar procesos de lectura y escritura. El *byte* alto se conoce como registro PCH, contiene los *bits* 8 al 12 <12:8> del PC y no puede leerse ni escribirse de manera directa.

La memoria de programa

El contador de programa de los microcontroladores PIC de rango medio puede direccionar un espacio en la memoria de programa hasta de 8K y 14 *bits* de longitud. Como todas las instrucciones son de palabra simple, un dispositivo que posea una memoria de programa de 8K x 14 *bits* tiene espacio para albergar 8K en instrucciones. Esto facilita al usuario establecer si un dispositivo tiene suficiente memoria de programa para una aplicación determinada. Esta aclaración es importante hacerla debido a que, al trabajar en un dispositivo de CPU CISC y Von Neumann, una memoria de programa de 8K puede llegar a tener sólo alrededor de 4K para almacenar instrucciones.

La memoria de programa en los microcontroladores PIC se divide en cuatro páginas de 2K cada una (0h-7FFh, 800h-FFFh, 1000h-17FFh, y 1800h-1FFFh), **figura 9.1**. Dependiendo de la referencia del microcontrolador PIC, se determina qué porciones de este mapa de memoria posee internamente. Para ello debe referirse a la hoja de datos del PIC específico con el fin de establecer su memoria disponible.

Para saltar entre las páginas de la memoria de programa, se deben modificar los *bits* altos del contador de programa (PC). Esto se hace escribiendo el valor deseado en un registro SFR denominado PCLATH (*Program Counter Latch High*). Sin embargo, si se ejecutan instrucciones secuencialmente el con-

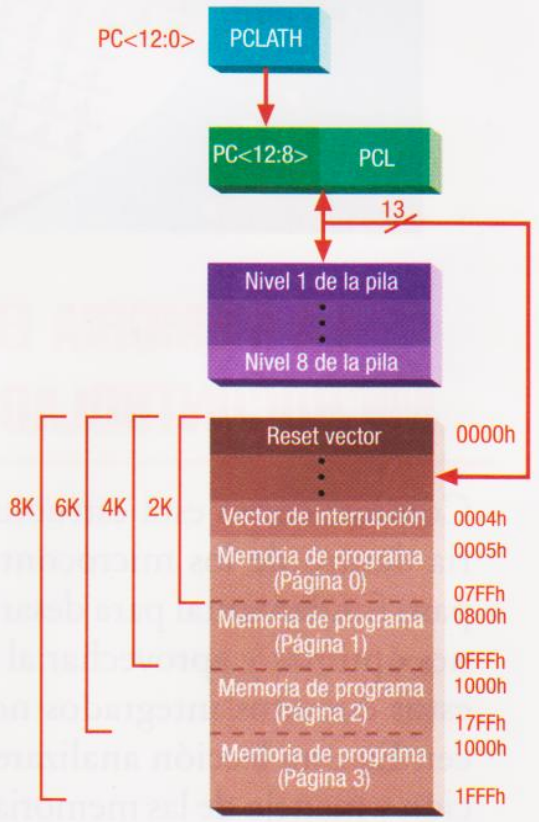


Figura 9.1 *Memoria de programa*

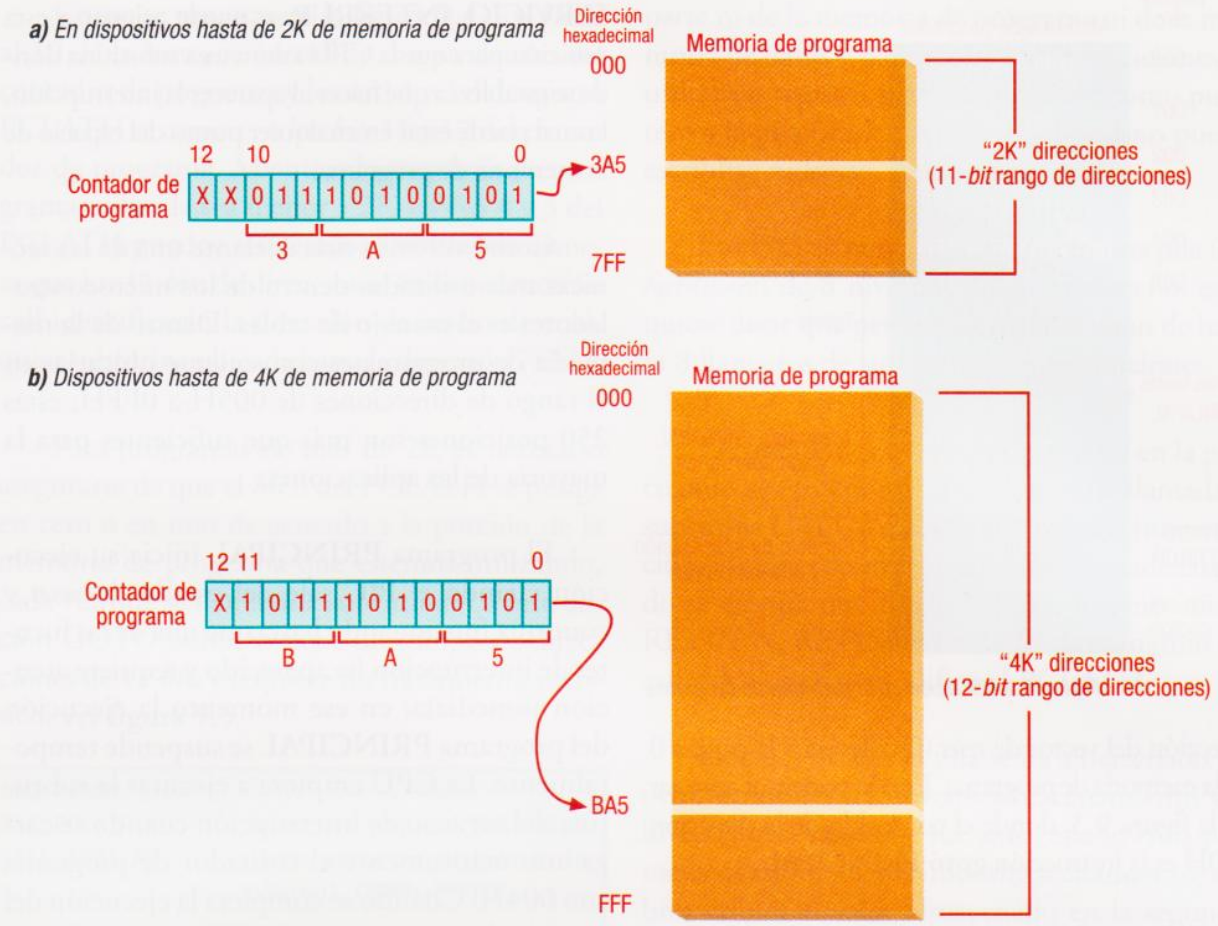


***Figura 9.2** Trabajo del contador de programa al acceder a la memoria de programa*

tador de programa puede llegar a sobrepasar los límites de la página sin ninguna intervención del usuario. Todas las actualizaciones que se hacen sobre el registro PCH afectan inmediatamente al registro PCLATH. Para dispositivos con menos de 8K el acceso a una posición por encima de las establecidas físicamente originará una sobreposición. Esto es, en un dispositivo de 4K acceder a la dirección 17FFh en realidad direcciona la 7FFh. Dispositivos de 2K o menos no requieren paginar la memoria de programa.

Una memoria de programa de 2K sólo necesita de un contador de programa de 11 *bits* para acceder a cualquier dirección (2^{11} = 2048 = 2K), **figura 9.2a**. Un microcontrolador con una memoria de programa de 4K necesita de un contador de programa de 12 *bits*, **figura 9.2b**.

Este último tipo de microcontroladores utiliza un contador de programa de 13 *bits*, permitiendo manejar hasta 8K de memoria de programa sin cambiar para nada la estructura de la CPU. Los dispositivos de 2K o 4K ignoran los *bits* más significativos del contador de programa durante los procesos de búsqueda en la memoria de programa.

Existen dos direcciones dentro de la memoria de programa que reciben un trato especial por parte de la CPU. Cuando la CPU inicia su operación desde su estado de *reset*, su contador de programa automáticamente toma el valor de cero. Esta dirección se denomina ***vector de reset*** debido a que a ella llegará la ejecución del programa cuando ocurra un estado de *reset*.

En este punto se borrará el contenido del registro PCLATH. Esto significa que cualquier salto hacia la

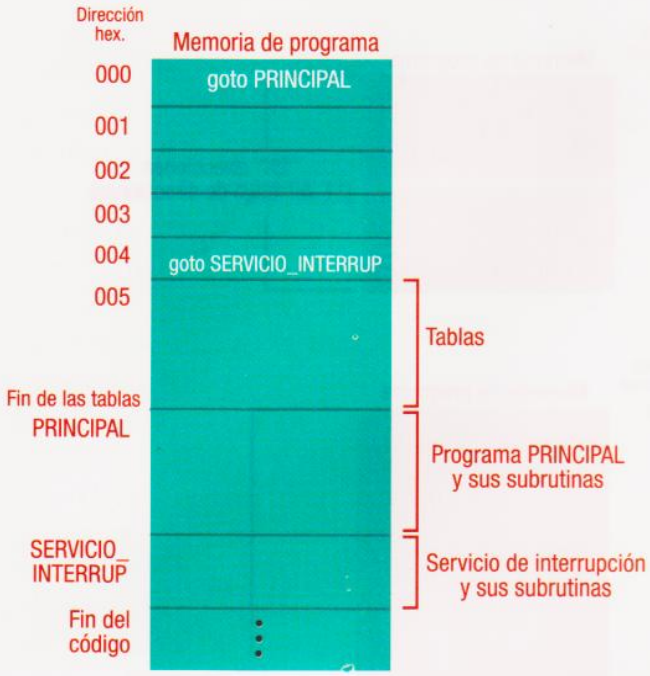


Figura 9.3 *Ubicación en la memoria de programa*

dirección del vector de *reset* (0h) llevará a la página 0 de la memoria de programa. Esto lo podemos apreciar en la **figura 9.3**, donde el contenido de la dirección 000H es la instrucción **goto PRINCIPAL**.

La segunda dirección especial es la 004H, la que se carga automáticamente en el contador de programa en el momento en que ocurre una interrupción. Esta dirección se conoce como ***vector de interrupción***. Cuando el PC toma la dirección del vector de interrupción, el registro PCLATH no se modifica. Como se muestra en la **figura 9.3**, una instrucción **goto SERVICIO_INTERRUP** se puede asignar a esta posición para que la CPU salte a una subrutina donde se establezca qué hacer al aparecer la interrupción, la cual puede estar en cualquier punto del espacio de la memoria de programa.

Como veremos más adelante una de las técnicas más utilizadas dentro de los microcontroladores es el manejo de tablas. Dentro de la memoria de programa es conveniente ubicarlas en el rango de direcciones de 005H a 0FFH; estas 250 posiciones son más que suficientes para la mayoría de las aplicaciones.

El programa **PRINCIPAL** inicia su ejecución cuando el PIC sale del estado de *reset* y continúa funcionando hasta que una de las fuentes de interrupción ha aparecido y requiere atención inmediata; en ese momento la ejecución del programa **PRINCIPAL** se suspende temporalmente. La CPU empieza a ejecutar la subrutina del servicio de interrupción cuando se carga automáticamente el contador de programa con 004H. Cuando se completa la ejecución del servicio de interrupción, la CPU retorna al mismo punto donde abandonó la ejecución del programa **PRINCIPAL**.

La escritura de programas se complica un poco cuando entre el código principal, tablas, subrutinas e interrupciones, se ocupa más de 2K.

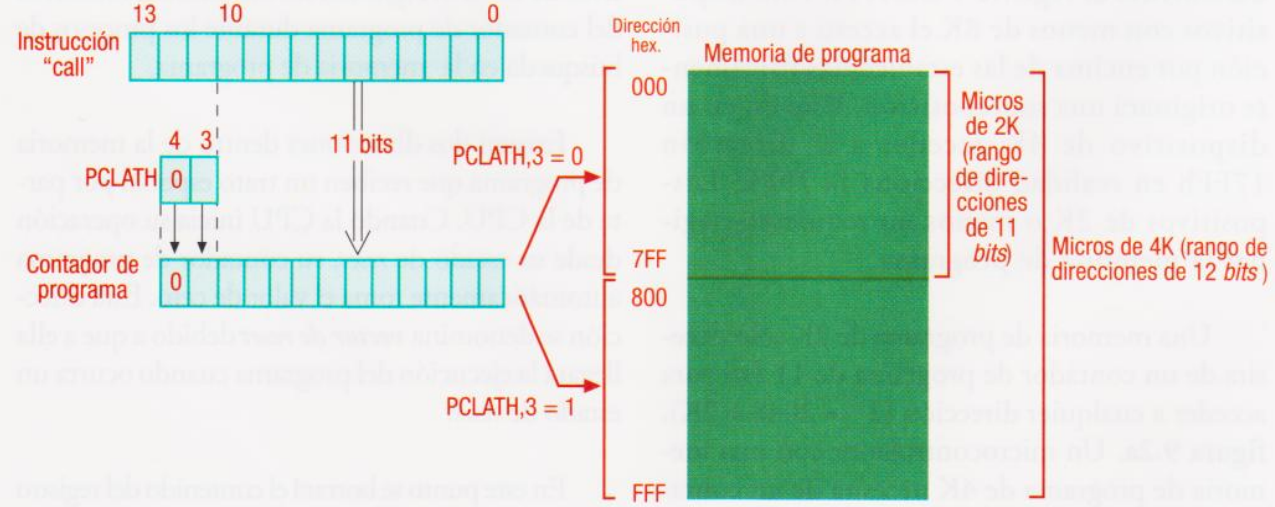


Figura 9.4 *Direccionamiento usado por llamado a subrutinas*

Como se muestra en la **figura 9.4**, los *bits* 0 al 10 de la instrucción **call** se cargan dentro del contador de programa y los *bits* 4 y 3 del registro PCLATH se cargan en los *bits* 11 y 12 del contador de programa. Mientras la memoria de programa utilizada sea menor a 2K los *bits* 4 y 3 del PCLATH pueden ser dejados en 00H, de manera que los 11 *bits* de direcciones en la instrucción **call** identificarán la dirección de inicio de cualquier subrutina ubicada hasta la dirección 7FFH.

Para programas de más de 2K es necesario asegurarse de que el *bit* 3 del PCLATH se ponga en cero o en uno de acuerdo a la porción de la memoria de programa que estemos utilizando, cada vez que se llama una subrutina. La instrucción GOTO también tiene un campo de direcciones de 11 *bits* y requiere un tratamiento idéntico, ver **figura 9.5**.

```
ORG 0x500
BSF PCLATH,3        ;Selecciono página 1(800h-FFFh)
CALL SUB1_P1        ;llamo subrutina ubicada en la
                    ;página1 (800h-FFFh)

ORG 0x900           ;
SUB1_P1:            ;subrutina llamada en la
                    ;página 1 (800h-FFFh)

RETURN              ;retorno al llamado de la
                    ;subrutina en la página 0
                    ;(000h-7FFh)
```

Figura 9.5 *Ejemplo del llamado de una subrutina en la página 1 desde la página 0*

La pila (*stack*)

La pila es un espacio de memoria que contiene la dirección de retorno antes de que ocurra un salto dentro de la ejecución de un programa. Al momento de ocurrir un salto dentro de un programa en la pila se almacena la dirección en donde se estaba llevando a cabo la ejecución, tomando el valor del contador de programa. De esta manera, la CPU cuando termina de ejecutar la subrutina, puede consultar en la pila la dirección de retorno para continuar con el desarrollo normal del programa. El espacio de la pila no forma parte ni de la memoria de programa ni de la memoria de datos y para acceder a sus posiciones se utiliza un registro especial conocido como puntero a la pila (*stack pointer*), en el cual no puede escribirse ni leerse.

Los PIC de rango medio poseen una pila (en *hardware*) de 8 niveles y 13 *bits* de ancho; esto quiere decir que permite la combinación de hasta 8 llamados de subrutinas e interrupciones.

El contador de programa es puesto en la pila cuando se ejecuta una instrucción de llamada a subrutina CALL o cuando aparece una interrupción. El valor presente en la pila es rescatado cuando se ejecuta una instrucción de retorno como RETURN, RETLW o RETFIE. En ninguno de estos dos casos se modifica el PCLATH.

Después de que la pila se ha almacenado en ocho ocasiones, el noveno almacenamiento sobreescribe el valor que fue almacenado en la primera ocasión. El décimo almacenamiento sobreescribirá el valor almacenado en la segunda ocasión, como se muestra en la **figura 9.6**. Es importante resaltar que no existe ningún *bit* en el registro de **ESTADO** que nos indique cuando se ha llenado la pila, de manera que es labor del programador verificar que dentro del programa no se hagan más llamadas a subrutinas que las soportadas por el microcontrolador.

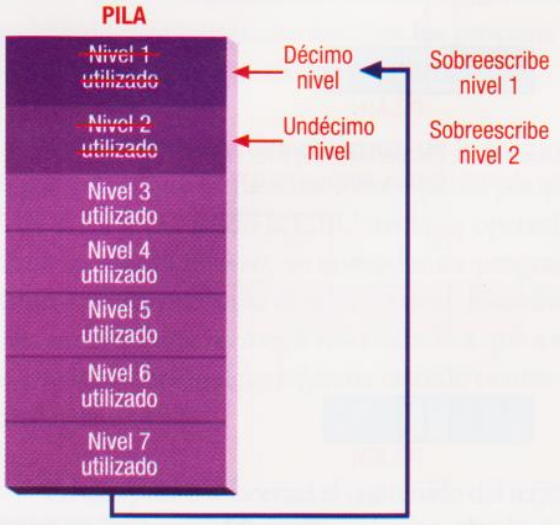


Figura 9.6 *Comportamiento de la pila*

Comportamiento del Contador de Programa (PC)

La **figura 9.7** muestra cuatro situaciones en donde podemos estudiar el comportamiento del contador de programa (PC). La situación 1 muestra como se carga el PC al escribir en el registro PCL (y por consiguiente el PCLATH<4:0> --> PCH), el PCH no puede modificarse (escribirse) directamente pero puede modificarse cuando varía PCL, y cualquier modificación que sufra PCH se refleja inmediatamente en el PCLATH. La situación 2 muestra como se carga el PC durante una instrucción GOTO (PCLATH<4:3> --> PCH) y vemos como los *bits* 3 y 4 del PCLATH (establecidos por el usuario) afectan el contenido del contador de programa. Como vimos anteriormente, de esta manera accedemos a páginas superiores en la memoria de programa. La situación 3 muestra como se carga el PC durante una instrucción CALL (PCLATH <4:3> --> PCH), cuando el PC se ha cargado en la cima de la pila (*stack*) y como, al igual que con la instrucción GOTO, el PCLATH afecta al contador de programa. La situación 4 muestra como se carga el PC durante una instrucción de retorno (RETURN), donde el PC es cargado con el valor que hay en la cima de la pila.

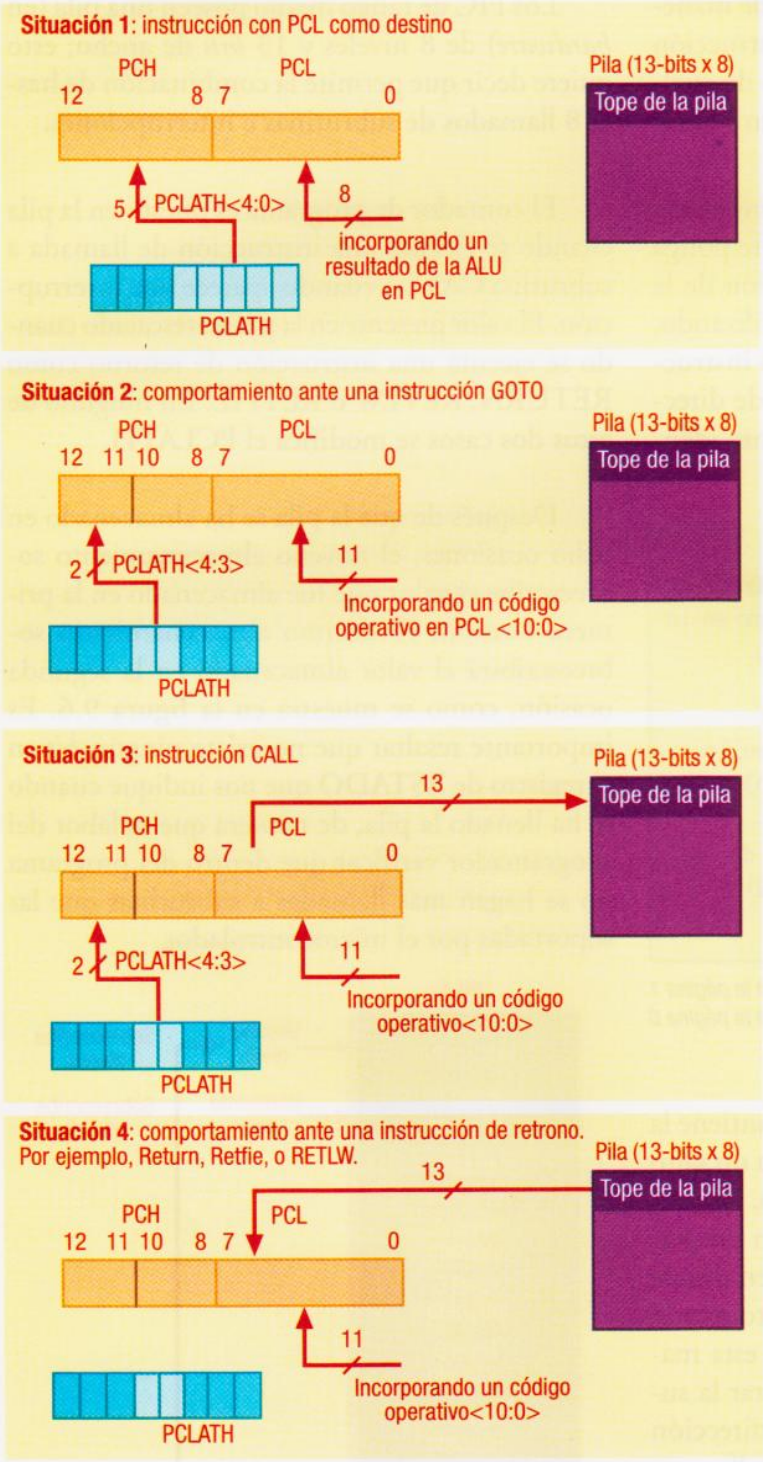


Figura 9.7 *Comportamiento del contador de programa*

En muchos programas en lenguaje ensamblador es necesario construir tablas de datos para distintas aplicaciones. Uno de los métodos más comunes para acceder a estas tablas es sumar un valor determinado al contador de programa (ADDWF PCL). Cuando invocamos una rutina de tabla el PC toma esa dirección inicial y al sumarle un valor fijo llega a la posición de la tabla indicada por dicho número. En un caso como éste, al sumarle un número de 8 *bits*, la tabla sólo podrá llegar hasta 256 posiciones de longitud.

Cualquier escritura que se haga sobre el contador de programa (PCL), originará que los 5 *bits* bajos de PCLATH se carguen en PCH.

La Memoria de Datos

Como habíamos mencionado anteriormente, esta memoria se divide en dos partes fundamentales: el área de registros con funciones especiales (SFR: *Special Function Registers*) y el área de registros de propósito general (GPR: *General Purpose Registers*).

Los registros de propósito general representan el área disponible para que el usuario almacene sus datos y desarrolle sus operaciones. Los registros con funciones especiales controlan la operación del dispositivo; por ejemplo, poseen los puertos y los registros de control para establecer el funcionamien-

Banco 0		Banco 1		Banco 2		Banco 3	
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	OPTION_REG	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	PORTB	186h
PORTC	07h	TRISC	87h	PORTF	107h	PORTF	187h
PORTD	08h	TRISD	88h	PORTG	108h	PORTG	188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
PIR2	0Dh	PIE2	8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh	OSCCAL	8Fh		10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h		91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPATAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRES	1Eh		9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
Registros de propósito general (2)	20h	Registros de propósito general (3)	A0h	Registros de propósito general (3)	120h	Registros de propósito general (3)	1A0h
			EFh		16Fh		1EFh
		Mapeado en el Banco 0 70h - 7Fh (4)		Mapeado en el Banco 0 70h - 7Fh (4)	170h	Mapeado en el Banco 0 70h - 7Fh (4)	1F0h
	7Fh		FFh		FFh		1FFh

Figura 9.8 Estructura general de la RAM

to de cada uno de los pines como entrada o como salida. Contienen también registros que permiten la entrada y salida de datos para la variedad de módulos del microcontrolador, como temporizadores, puertos seriales, conversores análogo-digitales, etc. También contienen *bits* de control que seleccionan el modo de operación de los módulos del microcontrolador, al igual que pueden habilitar o deshabilitar su funcionamiento.

Estructura en bancos

La memoria de datos (RAM) de los microcontroladores PIC se ha dividido en cuatro bancos. Cada uno de ellos contiene registros SFR y GPR. Para poder trabajar con cualquiera de estos registros, hay que estar ubicado dentro del banco correspondiente. La conmutación entre bancos requiere configurar los bits RP0 y RP1 del registro de estado (STATUS <7:5>) de acuerdo a lo mostrado en la **tabla 9.1**. La **figura 9.8** muestra un mapa general de la RAM para los microcontoladores PIC, que corresponde a una organización estándar para todos los dispositivos de esta familia. Los registros que se encuentran en negrilla siempre se encontrarán en cualquier dispositivo. Los demás registros dependen de la cantidad de periféricos que posea el PIC; por eso conviene mirar en las hojas de datos las características de cada dispositivo en particular.

En la estructura de los registros que se muestra en la **figura 9.8**, se puede apreciar que ellos poseen direcciones en 8 *bits* que van desde la dirección 00H hasta la FFH. El modo de direccionamiento directo utiliza sólo 7 *bits* de la instrucción para identificar la dirección de un registro, el octavo *bit* de la dirección debe venir de un *bit* (RP0) que selecciona el banco de registros con el cual se va a trabajar.

Banco accedido	Directo (RP1:RP0)	Indirecto (RP1:RP0)
0	0 0	0
1	0 1	
2	1 0	1
3	1 1	

Tabla 9.1 *Configuración de los bits RP0 y RP1*

En general, para mover valores de un registro hacia otro, el valor debe pasarse a través del registro de trabajo (W). Esto significa que para cualquier movimiento de datos entre registros se requieren dos ciclos de instrucción.

Toda la memoria de datos puede accederse directa o indirectamente. El direccionamiento directo requiere de la configuración de los *bits* RP0 y RP1. El direccionamiento indirecto requiere la utilización de un registro especial conocido como FSR (*File Select Register*) y el *bit* IRP (*Indirect Register Pointer*) que hace parte del registro de estado. Mediante ellos se puede acceder a los bancos 0,1,2 o 3 de la memoria de datos.

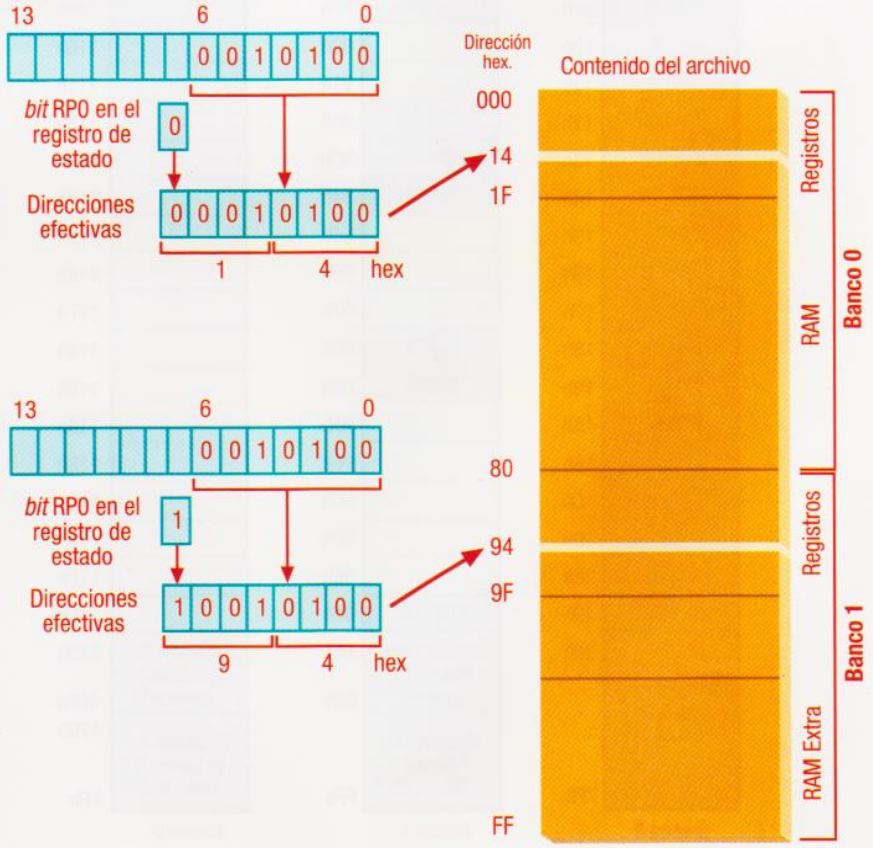


Figura 9.9a *Modo de direccionamiento directo*

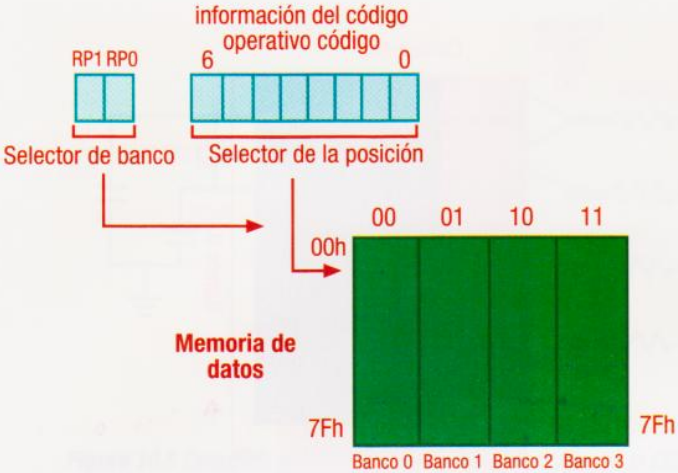


Figura 9.9b *Esquema general del modo de direccionamiento directo*

En la **figura 9.9a** se muestra la utilización del direccionamiento directo para acceder al registro ubicado en la dirección 14H o 94H, dependiendo del valor de RP0, y en la **figura 9.9b** se muestra un diagrama general de este tipo de direccionamiento, en el cual observamos como los 7 *bits* del código operativo de la instrucción (OPCODE) son utilizados para direccionar las posiciones de memoria y los *bits* RP0 y RP1 del registro de estado, pueden ser utilizados para direccionar o seleccionar los distintos bancos.

Cada banco se extiende hasta 7FH (128 *bytes*). Las posiciones más bajas de cada banco se reservan para los registros con funciones especiales (SFR). Debajo de los SFR se encuentran ubicados los registros de propósito general. Es de resaltar que algunos registros SFR del banco 0 se han puesto también en otros bancos (posiciones espejo) para facilitar su acceso. En la **figura 9.10** se muestra un mapa de memoria de datos correspondiente a algunos microcontroladores PIC de 18 pines, en donde podemos apreciar algunos registros quemanejan la ubicación en espejo. Por ejemplo el registro de estado (STATUS) se encuentra ubicado en las direcciones 03h y 83h. Direccionando cualquiera de estas dos posiciones podemos acceder y trabajar con dicho registro.

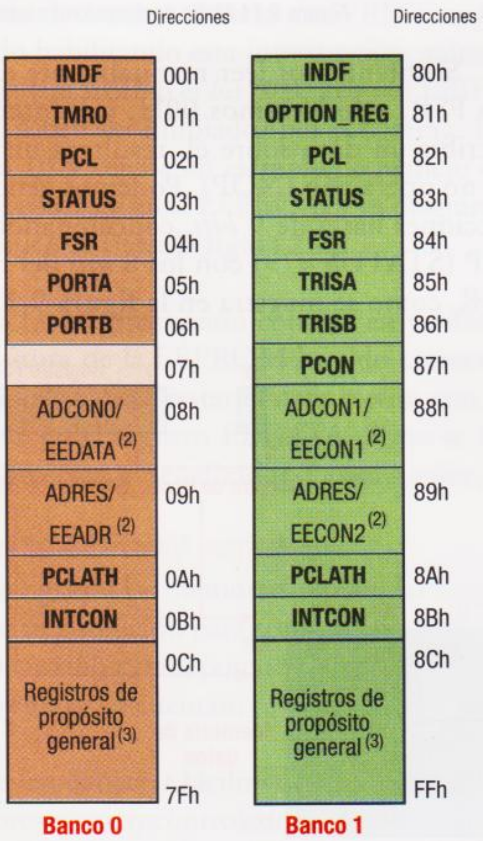


Figura 9.10 *Mapa de la memoria RAM de microcontroladores PIC de 18 pines*

Direccionamiento indirecto

Este modo de direccionamiento sobre la memoria de datos es apropiado en casos donde las direcciones de memoria que se manejan dentro de una instrucción no son fijas, es decir cuando se tiene el caso de que queramos trabajar con datos que se encuentren distribuidos a lo largo de varias posiciones de memoria. En este caso el direccionamiento indirecto nos brinda una manera rápida de acceder a todos esos datos, utilizando un registro especial (FSR) como apuntador a la posición en la memoria de datos que se quiere acceder. Como este apuntador es un registro en RAM, sus contenidos pueden modificarse dentro del programa y de esta manera apuntar a muchas posiciones distintas; esto puede ser muy útil cuando trabajamos con tablas de datos ubicadas en la memoria de programa; un ejemplo de este manejo lo podemos apreciar en el proyecto 5 de este curso titulado **Aviso Luminoso con Diodos LED.**

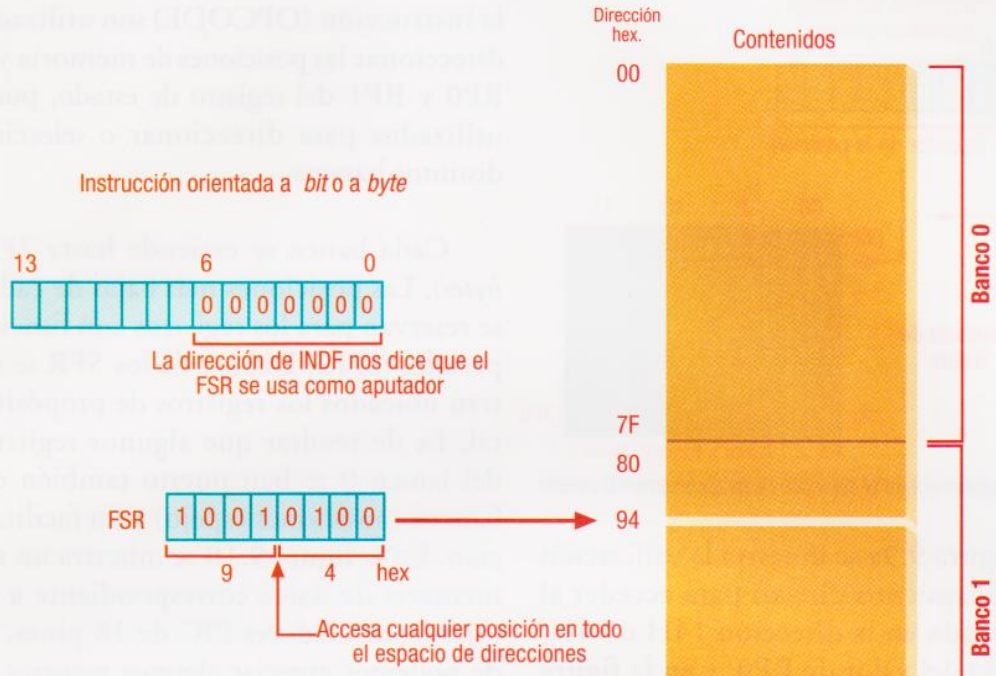


Figura 9.11 *Modo de direccionamiento indirecto*

En ese proyecto se escribió una serie de datos en diversas posiciones de la memoria de programa, los cuales debían ir mostrandose secuencialmente en la pantalla de diodos LED. Para resolver ese problema, simplemente se acudió al registro FSR para que hiciera un recorrido por todas esas casillas y pudieramos tomar el dato.

En este modo de direccionamiento, la dirección (en 8 *bits*) del registro que se va a acceder, se escribe en un registro especial (FSR) que sirve de apuntador a cualquier punto del mapa de memoria RAM. Después de haber escrito en ese registro, se puede consultar otro registro especial llamado INDF el cual accederá el registro usando el contenido del FSR como apuntador a la posición de memoria deseada por el operando, **figura 9.11**.

Cualquier instrucción que utilice el registro INDF puede acceder al registro apuntado por FSR.

Si intentamos leer manualmente el registro FSR, solo leeremos 00H, y si intentamos escribir un dato sobre él, resultará un estado de no operación (NOP). Podemos generar direcciones hasta de 9 *bits*, concatenando el *bit* IRP (STATUS <7>) con los 8 *bits* del registro FSR, como se muestra en la **figura 9.12**.

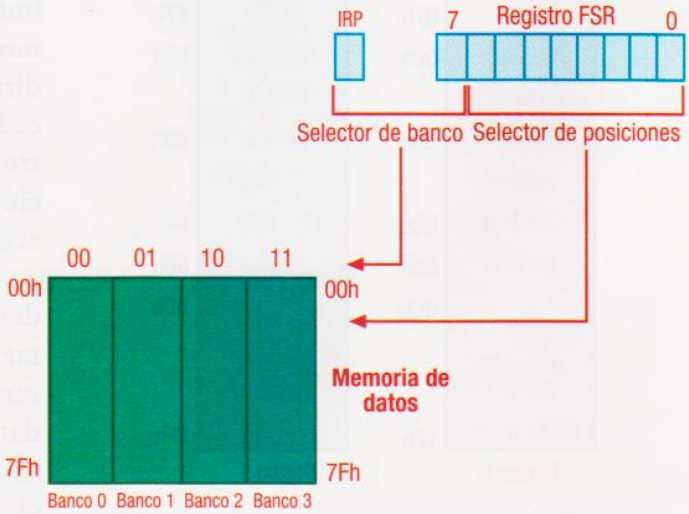
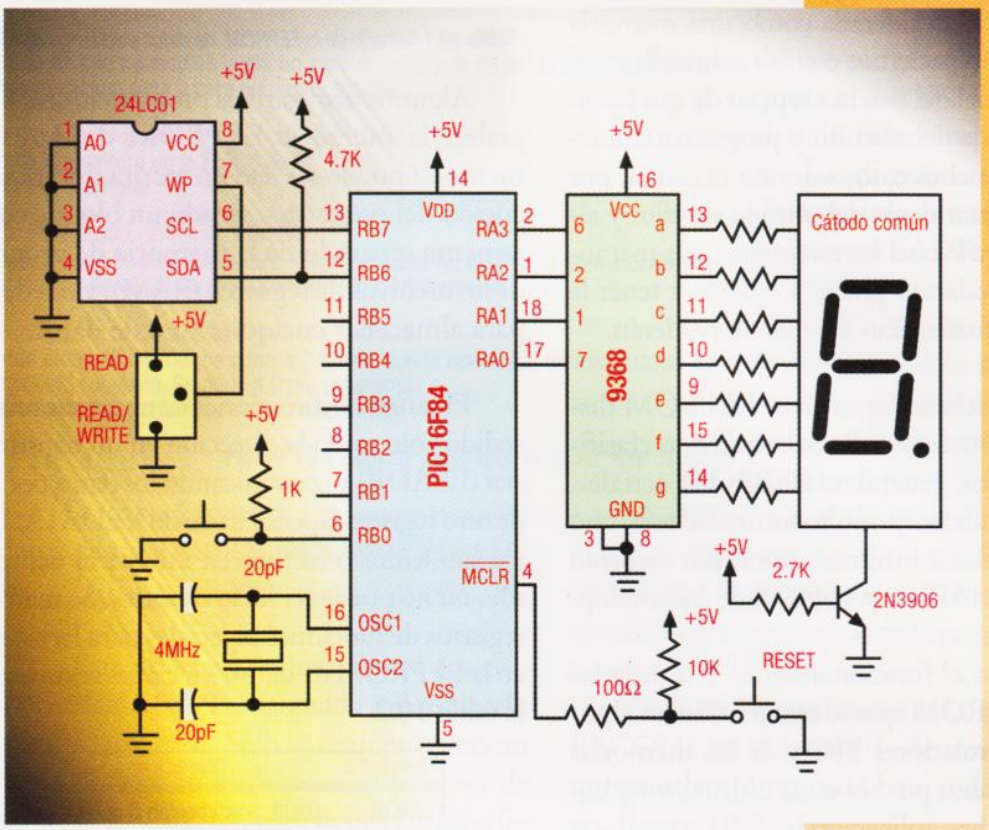


Figura 9.12 *Direccionamiento indirecto concatenando el bit IRP*



LA EEPROM DE DATOS EN LOS MICROCONTROLADORES PIC

En la lección anterior estudiamos los bloques de memoria RAM y ROM de los PIC. Ahora estudiaremos la memoria EEPROM de datos. Esta es una memoria que ocupa un bloque separado de las dos anteriores y que ha representado la solución a muchos problemas dentro del diseño electrónico. Al final hablaremos un poco de las memorias seriales que también ocupan un lugar relevante en una gran variedad de sistemas electrónicos.

Hemos visto las distintas tecnologías que tienen los microcontroladores para su memoria de programa: FLASH, EPROM, EEPROM, OTP, etc. Una vez se ha programado un microcontrolador, independiente de su tecnología, la memoria de programa puede ser considerada como una memoria ROM, ya que no podemos escribir sobre ella en el transcurso normal del sistema, a pesar de que luego podamos retirarla del circuito y programarla nuevamente. En muchos casos vamos a necesitar, por ejemplo, recolectar datos del mundo exterior y almacenarlos, para lo cual necesitaremos una memoria en la cual podamos grabar los datos y tener la certeza que permanecerán allí y no se perderán.

Existen muchos dispositivos EEPROM disponibles en el mercado. Estos se pueden clasificar en tres tipos generales: EEPROM seriales, EEPROM paralelas y microcontroladores con EEPROM de datos internas, como por ejemplo los PIC16F84, 16F870 y 16F87X de Microchip.

Analicemos el funcionamiento y uso de las memorias EEPROM que se encuentran en algunos microcontroladores PIC y de las memorias EEPROM seriales, por ser estas últimas muy comunes en muchas aplicaciones.

Memorias EEPROM internas

La memoria EEPROM de datos puede leerse y escribirse durante la operación normal de un sistema. En esta memoria se pueden almacenar datos y variables para que no se pierdan cuando se desconecta la alimentación del sistema. Analizando los requerimientos de la aplicación, puede ocurrir que esta memoria sea suficiente y no se tenga que recurrir a memorias externas.

Este tipo de memorias soportan hasta 1.000.000 de ciclos de escritura/borrado y pueden guardar la información sin alterarla por más de 40 años. Esta memoria no se encuentra en el espacio normal de archivos de registro; en lugar de eso, debe ser direccionada indirectamente a través de registros especiales. En la actualidad los microcontroladores PIC 16xxx ofrecen hasta 256 *bytes* de memoria EEPROM; ver **tabla 10.1**.

Tamaño de la EEPROM de datos	Rango de direcciones
64	0h - 3fh
128	0h - 7fh
256	0h - ffh

Tabla 10.1 *Tamaño de la EEPROM de datos y su rango de direcciones*

Algunos dispositivos programadores pueden grabar la memoria EEPROM de datos. Esta memoria puede ser leída o escrita durante la ejecución del programa, siendo un bloque completamente separado de la memoria de programa y de los archivos de registros (RAM), y puede usarse para almacenar cualquier tipo de datos.

El rango de direcciones de esta memoria es accedido colocando la dirección en un registro especial (EEADR) y comunicando los datos por medio de otro registro especial llamado EEDATA. Los datos son leídos o escritos en un *byte* al tiempo. Un sólo *bit* no puede ser leído o escrito. Se usan cuatro registros de funciones especiales para leer y escribir en la EEPROM de datos, los cuales se muestran en la **tabla 10.2**.

Registro	Dirección	
EEADR	0X09	->dirección 8 del banco 0
EEDATA	0X08	
EECON1	0X88	->dirección 8 del banco 1
EECON2	0X89	

Tabla 10.2 *Registros especiales para el manejo de la EEPROM de datos*

El tiempo de escritura es de aproximadamente 10 milisegundos y es controlado por el temporizador que hay dentro del circuito integrado. En realidad ese tiempo resulta muy largo para la velocidad del procesador, por este motivo existen varios *bits* en otro registro especial denominado EECON1 para supervisar la completa y correcta terminación del ciclo, **figura 10.1**. Analicemos los registros asociados al manejo de esta memoria:

EEADR. El registro de 8 *bits* EEADR es capaz de direccionar un máximo de 256 posiciones. El rango de direcciones utilizable lo podemos ver en la **tabla 10.1**.

EECON1

			EEIF	WRERR	WREN	WR	RD

RD: Lectura
1: Se pone en uno cuando se va a realizar un ciclo de lectura de la EEPROM. Luego pasa a cero automáticamente.

WR: Escritura
1: Se pone en uno cuando se inicia un ciclo de escritura en la EEPROM. Cuando se completa el ciclo pasa a cero automáticamente.

WREN: Permiso de escritura
1: Permite la escritura en la EEPROM
0: Prohibe la escritura

WRERR: Señalizador de error de escritura
1: Se pone en uno cuando una operación de escritura ha terminado prematuramente
0: La operación de escritura se ha completado correctamente.

EEIF: Señalizador de final de operación de escritura
1: Cuando este señalizador se pone a 1 indica que la operación de escritura se ha completado con éxito. Se pone a 0 por programa.
0: La operación de escritura no se ha completado.

Figura 10.1 *Registro de control de la EEPROM de datos*

EEDATA. Es un registro de 8 *bits* que se usa para pasar datos desde y hacia la memoria EEPROM de datos.

EECON2. Este no es un registro físico, sin embargo la dirección 0x89 es asignada a él. Al leerlo todos sus *bits* son cero. Sólo se emplea como un dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo. En él se escriben dos palabras de control y forman parte del ciclo de escritura.

EECON1. Este es el registro de control y tiene la función de controlar las operaciones en la EEPROM; su distribución de *bits* se muestra en la **figura 10.1**, y su descripción es la siguiente.

Los *bits* de control RD y WR inician la lectura y la escritura. Se deber tener en cuenta que estos *bits* sólo pueden ser colocados en uno (no se pueden colocar en cero) por *software* para que inicien el ciclo de escritura/lectura. Estos se limpian automáticamente al completarse un ciclo de lectura/escritura.

Una operación de escritura es válida cuando el *bit* WREN se pone en uno. WREN se limpia cada vez que se enciende el sistema. El *bit* WRERR se pone en uno cuando un ciclo de escritura se interrumpe por un *reset* debido al pin MCLR o al *watchdog*. Después de un *reset*, por medio de un programa, se puede chequear el *bit* WRERR y reiniciar el ciclo interrumpido de escritura. Las direcciones y los datos se mantendrán sin cambios en los registros EEADR y EEDATA.

Ciclo de lectura

Se inicia un ciclo de lectura colocando la dirección que se va a acceder en el registro EEADR y poniendo el bit RD = 1 en el registro EECON1. En el siguiente ciclo de instrucción, los datos estarán disponibles en el registro EEDATA y permanecerán allí hasta que se realice una nueva lectura o escritura en la EEPROM.

EEADR se localiza en el banco 0 (RAM) y EECON1 en el banco 1, de manera que se requiere seleccionar cada banco dentro del código, **figura 10.2**.

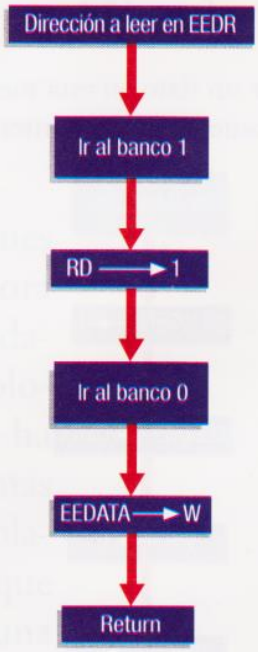


Figura 10.2 *Diagrama de flujo del ciclo de lectura en la EEPROM de datos*

Ejemplo de lectura de la EEPROM:

```
movlw  direccion     ; definir dirección a leer
movwf  eeadr         ; informar la dirección
                     ; al registro
bsf    status,rp0    ; banco 1
bsf    eecon1,rd     ; bit de lectura
bcf    status,rp0    ; banco 0
movf   eedata,w      ; datos hacia W
```

Ciclo de escritura

El ciclo de escritura comienza cargando en EEADR la dirección de la posición que se va a escribir y con el registro EEDATA el valor que se va a grabar. Después se debe añadir una secuencia de instrucciones que inician la escritura, en la cual participa el registro EECON2. Este registro, que en realidad no se halla establecido físicamente, sólo asume funciones de seguridad en el proceso cargándose en él dos valores concretos: 55H y AAH. El tiempo de estos pasos es un aspecto crítico, de manera que se recomienda que se siga esta secuencia. Las interrupciones deben deshabilitarse mientras se ejecuta este procedimiento. El *bit* WREN en EECON1 debe ponerse en uno para habilitar el WRITE. Esto previene escrituras accidentales en la memoria EEPROM de datos. El *bit* WREN debe ponerse en cero en todas las ocasiones, excepto cuando se ejecuta WRITE, ver **figura 10.3**.

Al escribir un dato en esta memoria, se borra automáticamente el dato anterior y se graba

Figura 10.3 *Diagrama de flujo del ciclo de escritura en la EEPROM*

```
;-MANEJO DE LA EEPROM--
        list    p=16c84
        radix   hex

;------------

; MAPEO DE MEMORIA
status          equ     0x03
portb           equ     0x06
eedata          equ     0x08
eeadr           equ     0x09
direccion       equ     0x0c
byte            equ     0x0d
trisb           equ     0x86
eecon1          equ     0x88
eecon2          equ     0x89
;-------------

;MAPEO DE BITS
rd              equ     0
wr              equ     1
wren            equ     2
rp0             equ     3
;-------------

                org     0x000
inicio          bsf     status,rp0      ;banco 1
                movlw   b'00000000'     ;salidas
                movwf   trisb
                bcf     status,rp0      ;banco 0
                clrf    portb
                movlw   B'00000010'
                movwf   byte            ;byte de
                                        ;prueba a grabar
                movlw   0x00
                movwf   direccion
                call    escritura
                bsf     status,rp0
escrcompleta    btfsc   eecon1,wr       ;byte
                                        ;escrito?
                goto    escrcompleta
                bcf     status,rp0
                call    lectura
                movwf   portb           ;mostrar
                                        ;dato en el puerto b
ciclo           goto    ciclo
;-------------
lectura         movf    direccion,w     ;
                                        ;tomar dirección
                movwf   eeadr           ;
                                        ;direccion hacia eeadr
                bsf     status,rp0      ;banco1
                bsf     eecon1,rd       ; leer
                                        ;dato de la EEPROM
                bcf     status,rp0
                movf    eedata,w        ;data-> w
                return                  ;retorna dato
                                        ;en w
;---------------
escritura       movf    direccion,w     ;toma
                                        ;direccion
                movwf   eeadr
                movf    byte,w
                movwf   eedata
                bsf     status,rp0
                bsf     eecon1,wren
                                        ;habilita escritura
                movlw   0x55
                movwf   eecon2
                movlw   0xaa
                movwf   eecon2
                bsf     eecon1,wr       ;WR -> 1
                bcf     eecon1,wren
                                        ; deshabilita
escritura
                bcf     status,rp0      ;banco0
                return
;----------------
                end
;FUSIBLES:
;MEMORIA DESPROTEGIDA
;WATCHDOG DESHABILITADO
;CRISTAL 4 MHZ
;POWER UP TIMER ON
;----------------
```

Figura 10.4 *Código de ejercicio de lectura y escritura en la EEPROM*

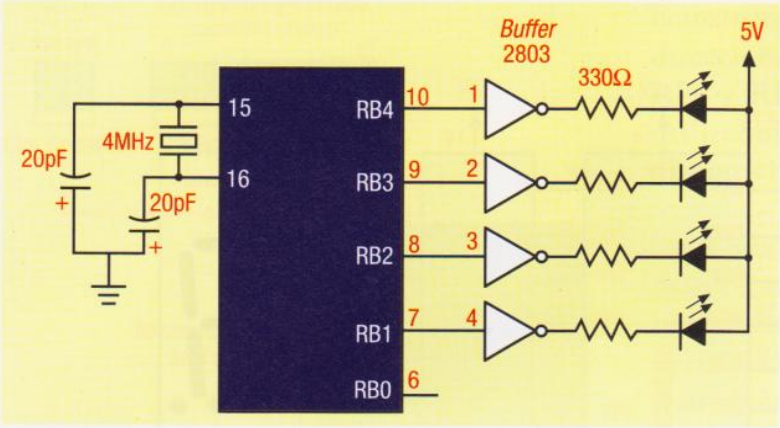


Figura 10.5 *Diagrama esquemático de lectura y escritura en la EEPROM de datos*

el nuevo. El tiempo de escritura es controlado por el temporizador (*timer 0*) del PIC. Este tiempo puede variar según las condiciones de voltaje y temperatura, al igual que entre diversos tipos de circuitos integrados.

Al completar el ciclo de WRITE, el *bit* WR se pone en cero automáticamente y se activa la bandera de escritura completa en EE. El programa puede detectar si el ciclo de WRITE se ha cumplido habilitando esta interrupción, mirando la bandera EEIF o el *bit* WR. Los *bits* EEIF y WREN deben ser limpiados por programa. En la **figura 10.4** vemos un ejemplo de manejo de lectura y escritura en la EEPROM y su diagrama esquemático se ve en la **figura 10.5**.

Una buena precaución consiste en verificar si la escritura de la EEPROM ha sido correcta, para lo cual se suele restar el dato escrito con el que existe en el registro EEDATA. Si no se ha producido error, el señalizador Z pasa a valer 1.

Memorias EEPROM seriales

La tecnología de las memorias seriales ha surgido como una solución para sistemas que requieren bajo tamaño y poca potencia. Entre sus principales ventajas se cuentan:

- Se pueden conectar fácilmente con microprocesadores o microcontroladores, incluso algunos de ellos tienen pines dedicados a esa labor.
- Transferencia de datos de manera serial, lo que permite ahorrar pines del microcontrolador; al igual que ocupan poco espacio y consumen poca potencia.

Cuando se ha decidido por el uso de este tipo de dispositivos, el paso siguiente es seleccionar uno de los dos protocolos posibles de comunicación con la EEPROM serial (2 o 3 hilos).

La elección de los dos hilos se debe hacer para aplicaciones que requieran *bus* I²C (este es un método de comunicación de datos serial que estudiaremos más adelante), inmunidad al ruido y disponibilidad limitada de pines. La elección de los tres hilos es conveniente en casos de requerimientos limitados de protocolos, un protocolo SPI (otro método de comunicación serial), alta frecuencia de reloj o en aplicaciones de 16 *bits*.

Bus de tres hilos. En este grupo se encuentran los dispositivos con la nomenclatura 93XXXX. En ellos se requieren cuatro pines.

Bus de dos hilos. La nomenclatura común es 24XXXX y 85XXXX, ver tabla 3. El protocolo I²C utiliza comunicación bi-direccional maestro/esclavo. Tanto el maestro como el esclavo pueden operar como transmisor o como receptor. La operación del *bus* debe ser manejada por el maestro, quien genera la señal de reloj serial y las condiciones de inicio de detención.

Estas memorias tienen un pin o terminal llamado SCL que recibe los pulsos generados por el maestro, y otro llamado SDA que maneja el flujo de datos bidireccionalmente. Este dispositivo no requiere de un pin habilitador, ya que en este esquema la transferencia de información sólo SE puede iniciar cuando el bus esté libre. Cada memoria tiene su dirección determinada mediante los pines A0, A1 y A2; en una red sólo responderá la memoria cuya dirección coincida con la dirección que va encabezando la trama de información.

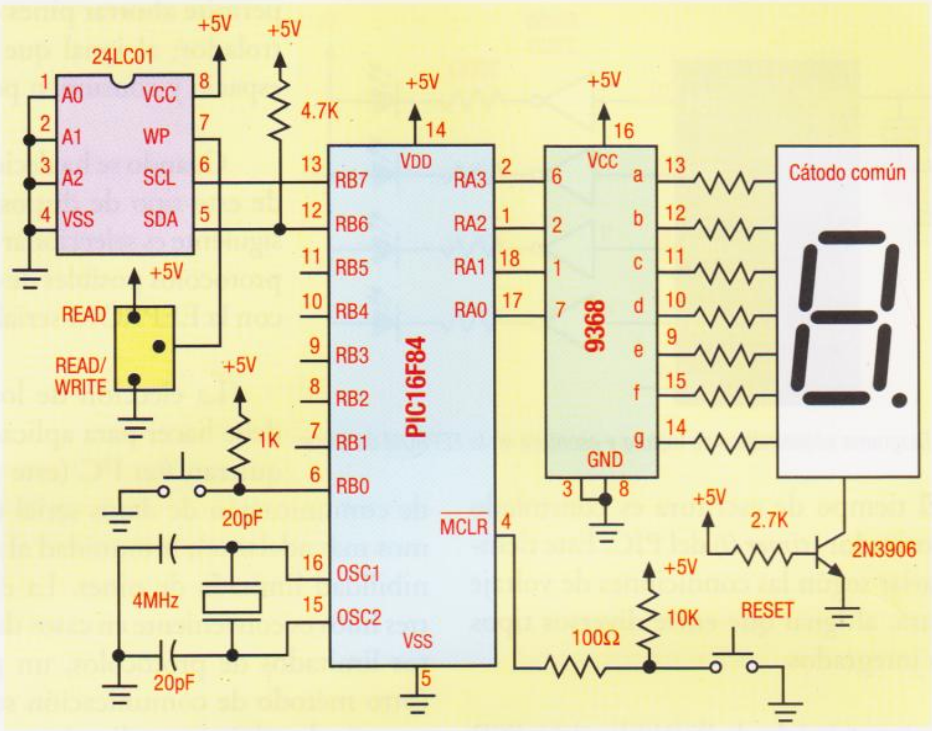


Figura 10.6 *Ejemplo del manejo de una memoria serial*

Cuando un microcontrolador desea entablar comunicación con la memoria, debe enviarle una serie de *bits* que lleven la siguiente información:

1.Se envía el *bit* de arranque
2. El código 1010 (propio de estas memorias)
3. La dirección del dispositivo (A2, A1, A0)
4. Un *bit* que indica que se desea escribir ('0') en la memoria

Después de esto, la memoria debe enviar un reconocimiento para informarle al microcontrolador que recibió la información. Dicha señal, llamada ACK, consiste en poner el *bus* en un nivel bajo (lo hace la memoria). Después el microcontrolador debe enviar los *bits* que corresponden a la posición de memoria que se quiere leer o escribir; nuevamente la memoria envía un reconocimiento. El paso siguiente depende de la operación que se vaya a ejecutar.

Ejemplo de aplicación

En este ejemplo se hará un contador de 0 a 9 con un interruptor pulsador y un *display* de siete segmentos, **figura 10.6**. El número que se muestra en el *display* se va a almacenar simultáneamente en una memoria 24LC01. En este caso, los pines de la dirección se conectaron a tierra, al igual que el pin WP. La resistencia de 4.7 Kohm conectada al pin SDA es necesaria dado que dicho pin tiene salida de colector abierto. El *display* se conecta al puerto A y el pulsador al pin RB0, **figuras 10.6** y **10.7**.

El programa lleva el control de la cuenta decimal y almacena en la memoria el mismo dato que se envía al *display*. El código de este ejercicio está disponible en nuestra página en la internet, en el enlace Curso práctico sobre microcontroladores.

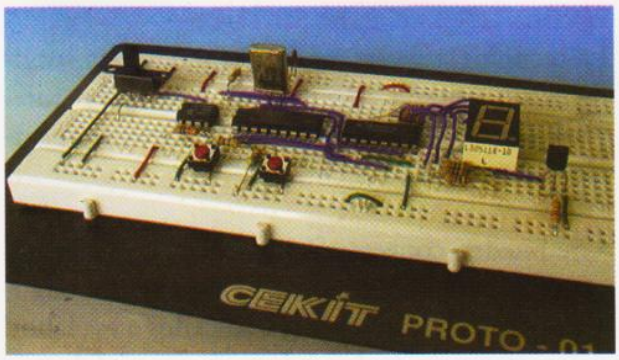
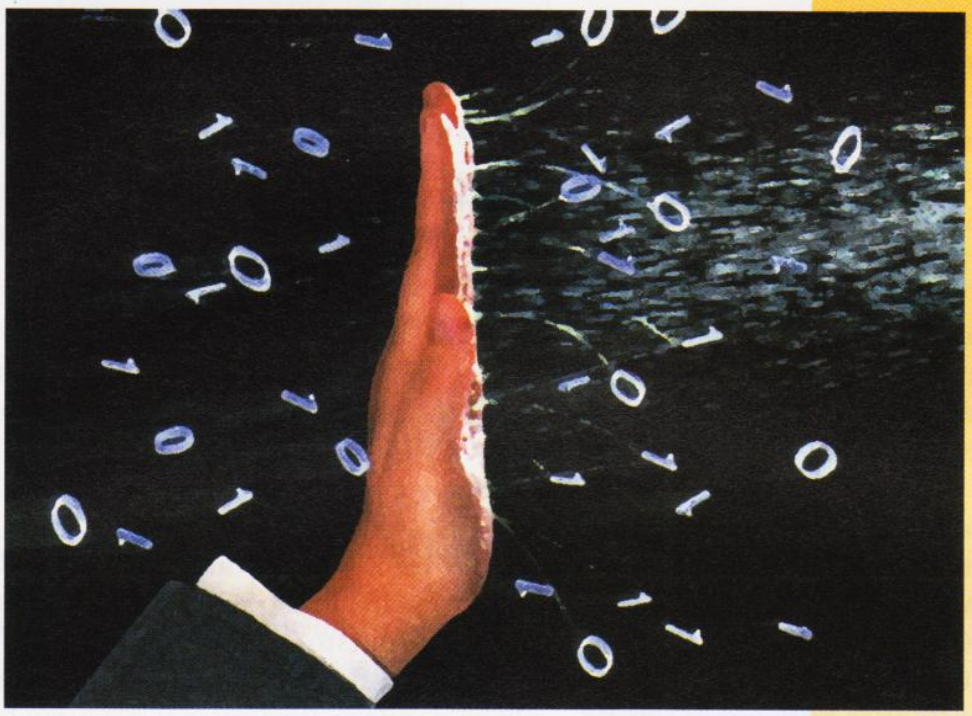


Figura 10.7 *Montaje del ejercicio*

LECCIÓN 11



INTERRUPCIONES EN LOS MICROCONTROLADORES PIC

Una de las herramientas más útiles que nos proporcionan los microcontroladores es la posibilidad de manejar interrupciones. Las técnicas disponibles para manejarlas deben entenderse muy bien para poder aprovechar al máximo los recursos y posibilidades que nos pueden brindar estos dispositivos.

Una interrupción es un evento que obliga a un sistema procesador a desatender por un momento la acción que se encuentre ejecutando dentro del desarrollo normal de un programa y llamar a una ***subrutina***, la cual usualmente se denomina **rutina** o **servicio de interrupción**.

Las fuentes de interrupción más utilizadas incluyen cambios de nivel, flancos de subida o de bajada en algunos pines y desbordes en los temporizadores, es decir, cuando se llenan completamente (FFH) y pasan de nuevo a 00H. La cantidad y el tipo de interrupciones dependen de cada referencia de microcontrolador; la lista es la siguiente:

- Por el pin INT (denominada interrupción externa). Es cuando al microcontrolador ingresa un flanco por un pin determinado (INT).
- Por desborde del temporizador TMR0
- Por cambio de nivel en el puerto B (pines RB7:RB4).
- Debida al módulo comparador
- Debida al puerto paralelo esclavo
- Debida al módulo USART
- De fin de conversión del módulo A/D
- Debida al módulo LCD
- De fin de escritura en la memoria EEPROM de datos
- Por desborde de los temporizadores *Timer1* y *Timer2*
- Debida al módulo CCP
- Debida al módulo SSP

Cuando aparece una interrupción, el procesador guarda en la pila (*stack*) la dirección de la instrucción en donde se encontraba y el control del programa se dirige hacia la dirección donde se encuentra la rutina de interrupción. Cuando ésta termina, recupera la dirección de la pila y vuelve al lugar exacto donde abandonó el programa.

En el mapa de memoria de programa de los microcontroladores PIC existe una dirección asignada (0004H) para ubicar la rutina. Cuando se origina un estado de *reset* (o estado de volver a inicializar el sistema), el ***Contador de Programa*** dirige la CPU a la dirección 000H; un programa que incluya interrupciones generalmente se estructura como se muestra en la **figura 11.1**.

```
      ORG   000H
      GOTO  MAIN
      ORG   004H
      GOTO  INT_SERV
MAIN:
      ;programa principal
      ;
INT_SERV:
      ;rutina de interrupción
```

Figura 11.1 *Ubicación normal de la rutina de interrupción*

Al retornar a la dirección donde se encontraba antes de la interrupción, se puede perder la información de algunos registros, como el de trabajo (W) y el de estado, por lo cual es responsabilidad del programador hacer una copia de ellos.

Tal vez la manera más adecuada de hacer esta operación es a través de un segmento de código dispuesto en el interior de un MACRO. En la **figura 11.2** se muestra un ejemplo en donde se almacenan y restauran los registros W y **ESTADO**. El registro W_TEMP debe definirse en todos los bancos y en la misma posición tomando como referencia la dirección base (por ejemplo, 0x70, 0x7F en el banco 0). Los pasos que se siguen para dar solución a este problema son los siguientes:

1. Almacenar W en el banco actual
2. Almacenar el registro ESTADO en el banco 0
3. Ejecutar la rutina de interrupción
4. Restaurar ESTADO (y el *bit* de selección de banco)
5. Restaurar W

Cada uno de los tipos de interrupción pueden habilitarse o deshabilitarse, el usuario es quien controla las fuentes de interrupción a través de un registro especial llamado INTCON, **figura 11.3**.

```
MOVWF W_TEMP              ;COPIAR W EN UN REGISTRO
                          ;TEMPORAL
;GUARDAR EL ESTADO NO ES TAN SENCILLO
;YA QUE MOVF ESTADO,W AFECTARA LOS BITS
;DEL MISMO REGISTRO ESTADO, PERO LA
;INSTRUCCION SWAPF NO LOS AFECTA
;POR ESO LA PODEMOS UTILIZAR

SWAPF ESTADO,1            ;GUARDANDO EL REGISTRO
SWAPF ESTADO,0            ;DE ESTADO
MOVWF ESTADO_TEMP

;LUEGO DEBEN SER RESTAURADOS.
SWAPF ESTADO_TEMP,1       ;RESTAURANDO ESTADO
SWAPF ESTADO_TEMP,0
MOVWF ESTADO
SWAPF W_TEMP,1            ;RESTAURANDO W
SWAPF W_TEMP,0
```

Figura 11.2 *Ejemplo donde se ilustra como guardar las copias de seguridad*

Es importante aclarar que los microcontroladores que solo cuenten con una interrupción de módulo periférico (por ejemplo el 16F84), han reemplazado el *bit* 2 (PEIE) por otros como el EEIE o el ADIE, que corresponden a interrupciones como la de la EEPROM de datos.

También hay que tener en cuenta que en algunos microcontroladores el *bit* 1 (INTF) no se encuentra contemplado. Por estas razones es conveniente verificar el mapa real del registro INTCON para cada tipo de microcontrolador.

Cada una de las interrupciones puede usarse sola o se pueden habilitar varias, dependiendo de la aplicación. En cualquier punto del programa, el usuario puede desactivar alguna de estas interrupciones borrando el *bit* correspondiente:

BCF INTCON, INTE; int. externa
BCF INTCON, RBIE; int. en RB
BCF INTCON, TOIE; int. por el TMR0

Cuando ocurre un estado de *reset* todos estos *bits* toman un valor de cero (limpiados).

Para habilitar cualquier interrupción, es necesario activar un *bit* (GIE) que se denomina el habilitador general o global. Es decir, las interrupciones se habilitan si se escribe BSF INTCON,GIE o se deshabilitan si se escribe BCF INTCON,GIE.

Registro: INTCON

GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

bit 7 — bit 0

Dirección: 0Bh
condición de *reset*: 0000000Xb

GIE: *Global Interrupt Enable* o Habilitador general de interrupciones.
0: deshabilita todas las interrupciones
1: habilita las interrupciones

PEIE: *Peripheral Interrupt Enable Bit* o Habilitación de interrupciones debidas a los módulos periféricos
0: la deshabilita
1: la habilita

TOIE: *TMR0 Interrupt Enable* o Habilitación de interrupción del temporizador TMR0.
0: la deshabilita
1: la habilita

INTE: *INT Interrupt Enable* o Habilitación de la interrupción INT.
0: la deshabilita
1: la habilita

RBIE: *RBIF Interrupt Enable* o Habilitación de la interrupción RBIF.
0: la deshabilita
1: la habilita

TOIF: *TMR0 Overflow Interrupt Flag* o Bandera de la interrupción por sobrefujo del TMR0.
Se coloca en 1 cuando el TMR0 pasa de 0FFh a 00h; ésta debe ser puesta a 0 por programa.

INTF: *INT Interrupt Flag* o Bandera de interrupción INT.
Se coloca en 1 cuando la interrupción INT (RB<0>) ocurre; ésta debe ser puesta a 0 por programa.

RBIF: *RB Port Change Interrupt Flag* o Bandera de interrupción por cambio en el puerto B.
Se coloca en 1 cuando una de las entradas RB<7:4> cambia; ésta debe ser puesta a 0 por programa.

Figura 11.3 *Registro INTCON*

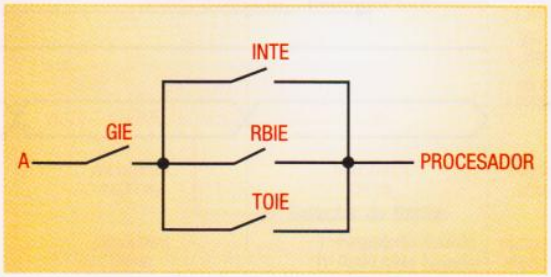


Figura 11.4 *Funcionamiento de la habilitación de las interrupciones*

Para entender este concepto, podemos hacer una analogía con un circuito serie, **figura 11.4**. Asumamos que GIE, RBIE, INTE y TOIE son interruptores en un circuito eléctrico y una corriente quiere circular desde el punto A hasta el procesador. Para que este fenómeno sea posible el interruptor GIE debe estar activado, al igual que cualquiera de los *bits* que habilitan a cada uno de los tipos de interrupción.

Cuando se ha configurado correctamente algún tipo de interrupción, existe una bandera asociada para cada tipo que se pone en "uno" cuando ella se ha presentado. Para determinar el tipo de interrupción basta leer cada una de esas banderas.

En el momento que ocurre una interrupción, el procesador limpia el *bit* GIE impidiendo que ocurra una nueva interrupción. Este *bit* se habilita de nuevo con la instrucción RETFIE cuando se retorna de la rutina de interrupción. El programador no debe preocuparse por esa habilitación ya que esto ocurre automáticamente.

Interrupción externa

La interrupción externa se genera cuando llega una señal externa al pin INT. El *bit* 6 del registro OPCION determina si esta interrupción la generará un flanco de subida o un flanco de bajada. Cuando aparece un flanco adecuado en el pin INT, la bandera INTF (*bit* 4 del registro INTCON) se habilita. Esa bandera debe desactivarse mediante un código en el programa dentro de la rutina de interrupción, antes de habilitar de nuevo esa interrupción.

Adicionalmente, si el dispositivo tiene otros periféricos que puedan originar interrupciones, entonces poseerá internamente algunos registros adicionales que permitan habilitarlas y otros registros donde se almacenen los *bits* que actuarán como banderas indicadoras. Dependiendo del dispositivo, los registros son:

- PIE1 y PIR1
- PIE2 y PIR2

Estos registros no mantienen una ubicación estándar para cada uno de sus *bits*. Cada tipo de microcontrolador posee algunas variaciones sobre la ubicación de ellos dentro del registro. Por esta razón, cuando hablemos de uno de los *bits* de estos registros no daremos su posición exacta. Dependiendo de la referencia del microcontrolador PIC, se debe buscar su posición en las hojas de datos.

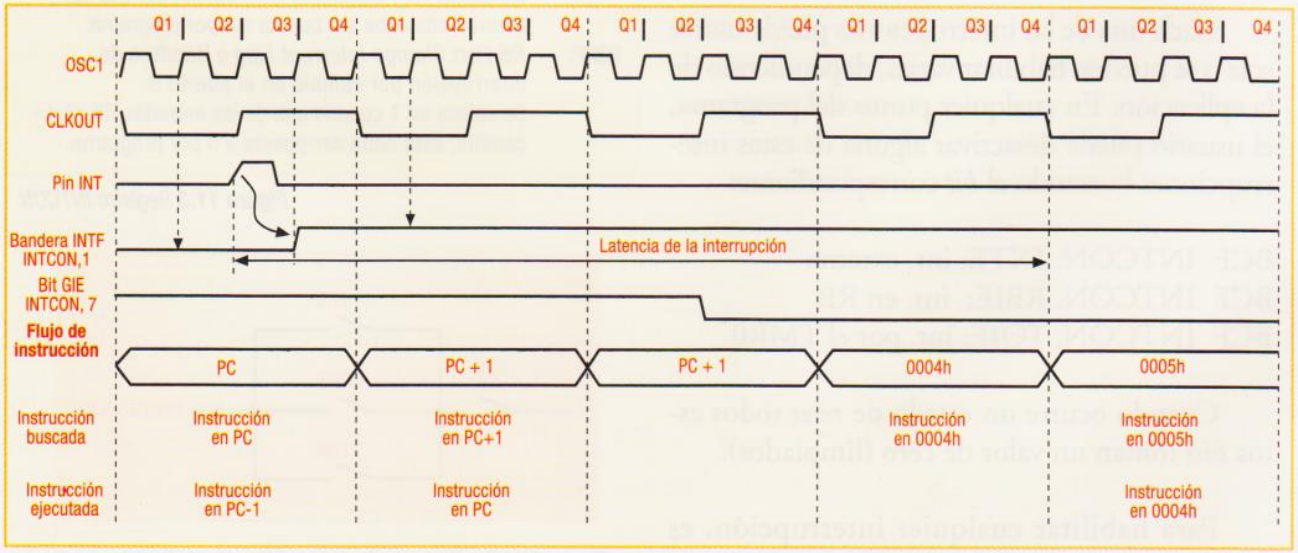


Figura 11.5 *Diagrama de tiempos del comportamiento del microcontrolador frente a una interrupción INT*

En el futuro es muy posible que se desarrollen módulos adicionales que puedan generar interrupciones, de manera que no sería raro encontrar más adelante pares de registros como el PIR3 y el PIE3. En apartados siguientes donde hablaremos sobre los temporizadores TMR1 y TMR2, notaremos la influencia de estos registros PIE y PIR.

```
;este programa cuenta el número de
;transiciones en la entrada RB0/INT
;y el resultado lo muestra en 4 LED
;colocados en RB4..RB7
      list p=16c84
A     #include <c:\p16c84.inc>
      _CONFIG 11H

CONSTANT BASE_VAR = 0CH
CONTADOR EQU     BASE_VAR+1
         ORG     000H
         GOTO    INICIO
         ORG     004H
         GOTO    SERV_DE_INT
INICIO:  BSF     STATUS,RP0
         MOVLW   1
         MOVWF   TRISB
         BCF     STATUS,RP0;BANCO 0
         CLRF    CONTADOR
         BCF     PORTB,4
         BCF     PORTB,5
         BCF     PORTB,6
         BCF     PORTB,7
         BSF     OPTION,INTDEG
         BCF     INTCON,INTF
                 ;BORRA BANDERA DE INTERRUPCION
         BSF     INTCON,INTE;
B                ;ENMASCARA INT. EXTERNAS
         BSF     INTCON,GIE
                 ;HABILITO INTERRUPCIONES
PT1:     SLEEP
         GOTO    PT1                    C
SERV_DE_INT:
         INCF    CONTADOR,F
         BTFSS   CONTADOR,0
         BCF     PORTB,4
         BTFSC   CONTADOR,0
         BSF     PORTB,4
         BTFSS   CONTADOR,1
         BCF     PORTB,5
         BTFSC   CONTADOR,1
         BSF     PORTB,5
D        BTFSS   CONTADOR,2
         BCF     PORTB,6
         BTFSC   CONTADOR,2
         BSF     PORTB,6
         BTFSS   CONTADOR,3
         BCF     PORTB,7
         BTFSC   CONTADOR,3
         BSF     PORTB,7
         BCF     INTCON,INTF
         ;LIMPIA LA BANDERA CORRESPOND.
         RETFIE

         END
```

Figura 11.6a *Código del ejemplo 1*

Existe un término conocido como **latencia de la interrupción**, que se define como el tiempo que transcurre desde que apareció la interrupción (la bandera se ha puesto en uno) hasta el momento en que la instrucción ubicada en la posición 0004H empieza a ejecutarse, este tiempo es de 3 TCY (ciclos de instrucción) para interrupciones sincrónicas (como por ejemplo las provenientes de un temporizador).

Para interrupciones no sincrónicas (normalmente las externas), como la INT o la del cambio en el puerto B, el tiempo de latencia está entre 3 y 3,75 TCY (ciclos de instrucción), **figura 11.5**.

En esta figura podemos observar el proceso que ocurre desde que se presenta una interrupción

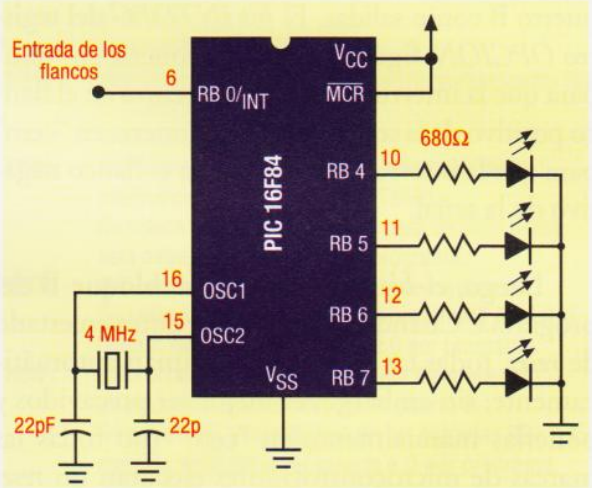


Figura 11.6b *Circuito de aplicación del ejemplo 1*

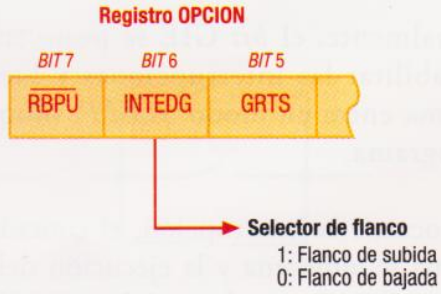


Figura 11.7 *Configuración del flanco que se va a leer por el pin INT*

originada por un flanco de subida por el pin INT. La bandera asociada se activa al siguiente ciclo Q y permanece en ese estado hasta que no sea borrada. Desde allí hasta que el microcontrolador llega a la dirección 0004H, han transcurrido de 3 a 4 ciclos de instrucción. El bit GIE que estaba en alto (de lo contrario no se hubiera podido haber leído la interrupción) es puesto en bajo automáticamente por la CPU para evitar leer otras interrupciones mientras ejecutamos la actual. La mejor manera de entender el manejo de las interrupciones es a través de un ejemplo.

Ejemplo de interrupción externa

El programa que observamos en la **figura 11.6a**, cuenta el número de transiciones positivas en el pin RB0/INT (pin 6) de un PIC16F84 y muestra el resultado de la cuenta en cuatro diodos LED colocados en las salidas RB4 a RB7, **figura 11.6b**. En la **parte A** del programa se configura el puerto RB0/INT como entrada y los demás pines del puerto B como salidas. El *bit INTDEG* del registro *OPCION*, **figura 11.7**, debe ponerse en "uno" para que la interrupción externa ocurra en el flanco positivo de la señal en RB0, o ponerse en "cero" para que la interrupción ocurra en el flanco negativo de la señal.

Luego, el *bit* **INTE** se activa, **bloque B** del programa. Cuando el procesador entra en estado de *reset*, todas las banderas se limpian automáticamente; sin embargo, es mejor ser precavidos y ponerlas manualmente en "cero" (no todas las marcas de microcontroladores ejecutan un *reset* tan limpio como los de Microchip).

Finalmente, el *bit* **GIE** se pone en uno para habilitar las interrupciones y luego el programa entra en modo *SLEEP*, **bloque C** del programa.

Al ocurrir la interrupción, el contador de programa se almacena y la ejecución del programa se direcciona a la posición 0004H, en donde nos ubicamos en la dirección de la rutina de interrupción.

Mientras nos encontramos ejecutando una rutina de interrupción, el *bit GIE* será deshabilitado por lo cual no se atiende a ninguna otra interrupción hasta que no salgamos de esa rutina (esto puede ser una desventaja ya que es posible que en un diseño especial deseemos esa característica).

En la rutina de interrupción, **bloque D**, el contador se incrementa y se visualiza en los diodos LED. En este caso se usó una técnica un poco dispendiosa para activar o desactivar los *bits* ya que en el siguiente ejemplo se usará el *bit* RB3 para otros propósitos.

El *bit* bandera asociado con la interrupción externa se limpia para permitir que el procesador se entere de que la petición de interrupción ha sido atendida. La instrucción *RETFIE* hace varias cosas. Ella habilita de nuevo el *bit GIE* y configura el contador de programa a su antiguo valor (el que tenía antes de la interrupción). El procesador entra en estado de *SLEEP* hasta que ocurra una nueva transición en el pin RB0/INT.

El estado de *SLEEP* es una posibilidad que brindan algunos microcontroladores, por el cual se entra a un modo de bajo consumo; este concepto lo estudiaremos en una lección posterior.

Interrupciones por cambio en un pin RB

El PIC16F84, al igual que muchos otros, permite establecer como fuente de interrupción el cambio de nivel en algunos pines determinados. En el PIC16F84 (al igual que en los 16C554/556), la interrupción se genera en los pines RB4, RB5, RB6 y RB7. Esta característica puede usarse como una buena técnica de ahorro en el consumo de potencia de un sistema. El procesador se puede poner en reposo (en modo *SLEEP*) y se despierta sólo cuando el

usuario cambia el estado de alguna de esas entradas. El procesador entonces ejecuta determinadas tareas y luego vuelve a reposo.

El procedimiento general para activar este tipo de interrupción es similar al que se sigue cuando se va activar la del *timer0* o la interrupción externa (INT). Se debe activar al *bit* **RBIE** del registro **OPCION** y luego activar el *bit* habilitador global de interrupciones (GIE). La operación interna del procesador, en este tipo de interrupción, consiste en leer el *nibble* alto del puerto B y hacer una copia; la interrupción se genera cuando la lectura del *nibble* alto del puerto B difiere de esa copia. Dentro de la rutina de interrupción es importante que se lea el puerto B para actualizar la copia. Para hacerlo, utilizamos las siguientes instrucciones:

```
BCF     INTCON,RBIF
MOVF    PORTB,W
```

Se observa que se limpia la bandera de interrupción en RB, pero el estado del *nibble* alto del puerto B aún sigue siendo diferente a la antigua copia, por eso el procesador inmediatamente activará la bandera *RBIF* y se activará de nuevo la interrupción. Por esta razón el código anterior es incorrecto, la manera adecuada es así:

```
MOVF    PORTB,W
BCF     INTCON,RBIF
```

Al ejecutar **MOVF PORTB,W**, la antigua copia del *nibble* alto toma el estado actual y luego se limpia la bandera. De esta manera, la siguiente interrupción ocurrirá cuando el *nibble* alto del puerto B cambie de nuevo.

Problemas comunes al trabajar con interrupciones

Ahora analizaremos algunos inconvenientes que se pueden presentar al trabajar con interrupciones, con lo cual posiblemente ahorremos muchas horas de trabajo.

Problema 1: ***al desactivar las interrupciones ¿cómo asegurarse que realmente lo están?***

Si tenemos un programa donde debemos atender las interrupciones mientras se ejecutan varias tareas (TAREA_1), pero no deseamos que se atiendan interrupciones mientras se ejecutan otras tareas (TAREA_2), se puede pensar en el código de la **figura 11.8.**

De manera inesperada fuimos interrumpidos mientras ejecutábamos el código de la TAREA_2. Por alguna razón la instrucción **BCF INTCON,GIE** no está haciendo su trabajo. ¿Que pasó entonces? ¿ocurrió una interrupción mientras se estaba ejecutando **BCF INTCON,GIE**?

Cuando ocurre una interrupción, la instrucción que se está ejecutando en el momento termina de cumplirse y luego el programa se dirige a la dirección 0004H donde se encuentra la rutina.

Al ejecutar **RETFIE** el programa retornará a la primera línea de TAREA_2 con las interrupciones aún habilitadas. Por eso el *bit GIE* debe limpiarse, pero debido a que la ejecución de la rutina de interrupción causa que dicho *bit* se ponga en uno, cuando ejecutamos la rutina TAREA_2 las interrupciones aún se encuentran habilitadas. La solución se muestra en la **figura 11.9.**

```
LAZO:

TAREA_1:
      BSF   INTCON,GIE
      ;SE REALIZAN TAREAS
      ;ACEPTANDO INTERRUPCIONES
      ;LAS TAREAS VAN EN ESTA
      ;PARTE DEL PROGRAMA
      BCF   INTCON,GIE
      ;AHORA SE DESHABILITAN TODAS LAS
      ;INTERRUPCIONES

TAREA_2:
      ;SE REALIZAN VARIAS
      ;TAREAS SIN ATENDER INTERRUPCIONES
      ;ESAS TAREAS VAN EN ESTA PARTE DEL
      ;PROGRAMA

      GOTO  LAZO
```

Figura 11.8 *Código del problema 1*

```
LAZO:

TAREA_1:
      BSF   INTCON,GIE
      ;REALIZA VARIAS TAREAS ACEPTANDO
      ;INTERRUPCIONES

APAGO_INTERRUPS:
      BCF   INTCON,GIE;APAGO EL GLOBAL
      BTFSC INTCON,GIE
      GOTO  APAGO_INTERRUPS

TAREA_2:
      ;REALIZA VARIAS TAREAS
      ;SIN ATENDER INTERRUPCIONES

      GOTO LAZO
```

Figura 11.9 *Solución al problema 1*

Notemos que el *bit* GIE se ha limpiado y verificado, para estar completamente seguros de que se ha limpiado correctamente. El proceso debe repetirse cuantas veces sea necesario para que esté deshabilitado.

Así, si una interrupción ocurre mientras se ejecuta **BCF INTCON,GIE**, la rutina de servicio de interrupción activará ese *bit*, pero el programa continuará con la instrucción **BTFSC INTCON,GIE** donde detecta que **GIE** está activado y por lo tanto se debe hacer algo para apagarlo. De esta manera el programa sólo llegará a TAREA_2 después de que se ha constatado que el *bit* **GIE** ha sido limpiado.

Problema 2: ***bancos de registros: ¿qué pasa cuando estamos ejecutando un programa donde necesitemos atender interrupciones cuando estemos en el banco 0 ó en el banco 1?.***

Al inicio de la rutina de interrupción, los registros **W** y **ESTADO** se almacenan en **W_SEGUR** y **ESTADO_SEGUR**. Asumamos que estas variables son 20H y 21H. Si la interrupción aparece cuando nos encontramos en el banco 0, ellos se almacenan en las direcciones 20H y 21H. Pero si la interrupción ocurre cuando nos encontramos en el banco 1, esos registros se almacenan en las direcciones A0 y A1H.

Nuestro servicio de interrupción debe conmutar al banco 0 y luego restaurar **W** y **ESTADO**, pero en realidad recuperamos los valores que se buscaron en A0 y A1. En el caso particular de un 16F84, esto no es un problema grande, ya que la RAM del usuario se divide en dos bancos los cuales uno es espejo del otro. Esto es, 20H y A0H de hecho están en la misma posición. El problema real es cuando se utilizan microcontroladores donde los bancos no son espejos entre ellos. En ellos 20H y A0H son registros completamente diferentes.

Para solucionar ese problema, se pueden permitir interrupciones sólo cuando nuestro programa se encuentra trabajando en el banco 0. De hecho, usualmente utilizamos el banco 1 para configurar los registros tri-estado lo cual toma muy poco tiempo, y en muchos casos, deshabilitar las interrupciones durante este lapso de tiempo no nos trae ningún problema, **figura 11.10**.

```
....
            ;voy al banco 1
APAGO_INTERRUPS:
      BCF   INTCON,GIE
            ;DESHABILITO
            ;INTERRUPCIONES
      BTFSC INTCON,GIE
      GOTO  APAGO_INTERRUPS
      BSF   STATUS,RP0;BANCO 1
            ;TRABAJO CON TRISA O TRISB
      BCF   STATUS,RP0;BANCO 0
      BSF   INTCON,GIE
            ;AHORA HABILITO
            ;INTERRUPCIONES
```

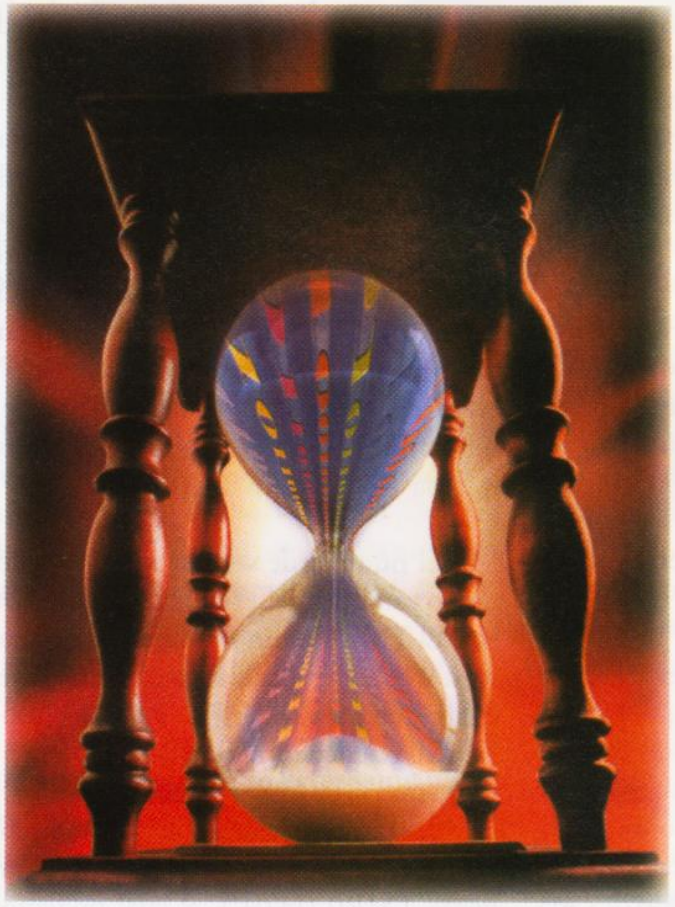
Figura 11.10 *Problema 2*

Comentarios finales

Todos las observaciones que aquí se analizaron, pueden resultar muy obvias después de ver su solución. Pero si no la conocemos seguramente gastaremos varios días averiguando que pasó.

Cuando se desarrollan productos (sobre todo de calidad comercial) es muy importante entender los problemas que se puedan presentar. En ese proceso de entendimiento se pueden emplear varios días (meses, años o tal vez nunca). Cuando no se logra un entendimiento total del problema, es cuando empiezan a fallar los productos con el transcurrir del tiempo.

LECCIÓN 12



LOS TEMPORIZADORES EN LOS MICROCONTROLADORES PIC

Como hemos visto en lecciones anteriores, dentro de los microcontroladores existen unos módulos que podemos utilizar para funciones exactas de medición de tiempo, llamados temporizadores. En esta lección analizaremos estos elementos internos presentes en los microcontroladores PIC. En la sección de programación, desarrollaremos ejemplos prácticos para el manejo de estos importantes elementos.

En la familia de los microcontroladores PIC aparecen tres módulos temporizadores denominados: *Timer0* (TMR0), *Timer1* (TMR1) y *Timer2* (TMR2). Posiblemente el *Timer1* sea el más versátil ya que puede usarse para monitorear el tiempo entre las transiciones de una señal ocurridas en un pin de entrada o para controlar de manera precisa el tiempo de transición en un pin de salida. El *Timer0* puede utilizarse para contar eventos externos (transiciones de señales) o generar interrupciones cuando ha ocurrido un número deseado de eventos.

Existe la manera incluso, a través de un módulo interno denominado prescalador, de leer entradas a una rata de velocidad de hasta 50 Mhz, es decir, que podemos lograr mucha más velocidad que cuando leemos normalmente entradas por los pines; este ejercicio lo analizaremos en el proyecto del frecuencímetro incluido en este curso.

El *Timer2* tiene mucha más libertad en su funcionamiento y además de temporizador, puede usarse para manejar el período de una señal de salida en formato de modulación por ancho de pulso (PWM). En este tipo de señal, la salida de un pin permanece en estado lógico alto durante un porcentaje determinado de cada periodo.

El porcentaje se controla escribiendo en el registro CCP (***Capture/Compare/PWM***), y el periodo de la señal de salida se controla por el *Timer2*. Este tipo de señales es muy utilizado en el control de motores, luces, etc.; el módulo CCP lo analizaremos en detalle en la siguiente lección.

Temporizador 0 (*Timer0*: TMR0)

Este temporizador puede entenderse como un registro de 8 *bits*, con el cual podemos hacer operaciones de lectura o escritura en cualquier momento. Puede incrementarse automáticamente ya sea por medio de los ciclos internos del reloj o por medio de pulsos externos que entran por un pin específico del microcontrolador y nos sirven para crear bases de tiempo muy precisas o para contar eventos externos.

Por este motivo a este módulo se le ha denominado **temporizador/contador**, ya que puede configurarse para que sirva de temporizador o que actúe como contador de eventos.

Cuando el *Timer0* inicia su operación, se incrementa con cada ciclo de máquina o con cada pulso externo introducido por el pin RA4/TOCKI; la elección de cual va a ser la fuente de señal se configura en el registro OPCIÓN. El registro TIMER0 (de 8 *bits*) se va llenando hasta llegar a FF; si en ese momento llega otro incremento pasa a 00 y justo ahí puede generar una interrupción, con lo cual podemos generar una base de tiempo confiable.

El número de veces que el contador llega a cierto valor puede ser contado usando otro registro. El TMR0 es un módulo autónomo, de manera que el procesador puede estar ejecutando cualquier tarea mientras que el temporizador TMR0 se encuentra trabajando.

Las posibles salidas que podemos obtener del TMR0 son dos: la primera, es la lectura del registro TMR0 (H'01'), el cual podemos leer en cualquier momento y tomar decisiones sobre esa lectura, y la segunda, es generar una interrupción por desborde del registro, es decir, cuando se llena tomando el valor de FF y pasa a 00.

El registro OPCIÓN (*OPTION*) cuya estructura la estudiamos en la lección 8 de este curso, es el que utilizamos para configurar el funcionamiento de este módulo de temporización. El *bit* 5 (TOCS) lo configuramos para que el TMR0 funcione como temporizador (se incremente con el reloj interno de la máquina) o como contador (se incremente con los pulsos externos leídos por un pin determinado del microcontrolador llamado TOCK). Para detectar cuál es el pin que corresponde a esta función en cada tipo de micro, debemos referir-

Figura 12.1 *Esquema en bloques del TMR0*

nos a las hojas de datos. Con un cero (0) en este *bit* el TMR0 funciona como temporizador, y con un uno (1) funciona como contador.

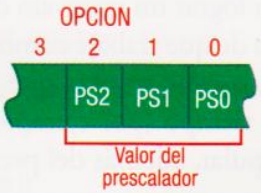
Hay que tener en cuenta que si el TMR0 se ha configurado como contador, él sólo funcionará con flancos, no con niveles; es decir, que sólo contará las transiciones de cero a uno (flanco de subida), o de uno a cero (flanco de bajada). Por esta razón se le debe decir al TMR0 qué transición queremos que cuente a través del *bit* 4 del registro OPCIÓN denominado TOSE; poniendo este *bit* en cero (0), el TMR0 contará todos los flancos de subida y poniéndolo en uno (1), el TMR0 contará todos los flancos de bajada.

Para modificar los contenidos del registro OPCIÓN, existen dos opciones: la primera consiste en ubicarnos en el banco correspondiente, cargar W con el número que llevaremos a OPCIÓN y pasar los contenidos de W a dicho registro mediante la instrucción **MOVWF OPCION**, luego se debe retornar al banco inicial para continuar trabajando normalmente. La segunda no requiere movernos de banco, simplemente consiste en dar la instrucción especial **OPTION**, la cual moverá inmediatamente los contenidos que tengamos en ese momento almacenados en el registro W al registro OPCIÓN.

La fuente de señal hacia el TMR0 puede ser sometida a un proceso de división de frecuencia a través de un módulo denominado el prescalador, **figura 12.1**. En este módulo la frecuencia de entrada es dividida por un valor determinado por los *bits* 0, 1 y 2 del registro OPCIÓN, que corresponden al valor PS0, PS1 y PS2, según la **tabla 12.1**. La cuenta o la temporización del TMR0 puede hacerse sin la intervención del prescalador; para hacerlo, debemos poner un uno (1) en el *bit* 3 del registro OPCIÓN: PSA.

Para tener una lectura confiable cuando vamos a trabajar con una señal de reloj externa, es mejor tener en cuenta las siguientes recomendaciones:

1. Si no se va a utilizar el prescalador, la entrada debe permanecer en alto (o en bajo) durante al menos dos (2) períodos de oscilación del reloj del microcontrolador (Tosc).
2. Si se va a utilizar el prescalador, el periodo de entrada debe ser de al menos cuatro (4) Tosc dividi-



PS2	PS1	PS0	Dividir por
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

Tabla 12.1 *Valores configurados en el registro OPCIÓN para el prescalador*

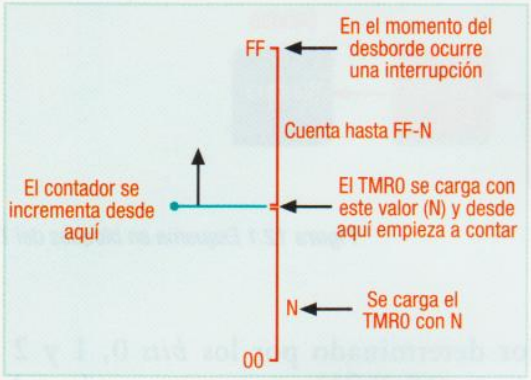


Figura 12.2 *Esquema del funcionamiento del TMR0*

dos por el valor del prescalador, y los niveles altos y bajos deben ser mayores a 10ns de duración.

Si existe una operación de escritura sobre el TIMER0, el incremento se inhibe por los siguientes dos ciclos de instrucción. Esto puede compensarse ajustando el número cargado en el registro.

El prescalador

Como ya se mencionó, el prescalador es un módulo interno que divide la frecuencia de reloj de entrada al temporizador. Para ejecutar esta tarea utiliza como factor de división el número configurado en OPCIÓN en los *bits* 0, 1 y 2. De esta manera el usuario puede jugar con este factor de división para lograr un intervalo de tiempo preciso (en el caso de que trabaje como temporizador). En este punto cabe resaltar que para ajustar un intervalo de tiempo (por ejemplo 1 segundo) se puede manipular, además del prescalador, el tipo de oscilador externo que más se ajuste; en el proyecto Reloj Digital con PIC presente en este curso observaremos como se creó un intervalo exacto de un segundo mediante un cristal de 1.87MHz.

La única manera de no prescalar el temporizador es asignarlo al perro guardián (***watchdog***). Se trata de un registro de 8 *bits* que cumple con una labor muy específica (como veremos más adelante) al cual podemos asignar este factor de división. Algunos autores mencionan que cuando se asigna al TMR0 funciona como prescalador y cuando se asigna al *watchdog* funciona como postescalador. Sin embargo en muchas fuentes bibliográficas se conoce simplemente como prescalador sin distinguir si hace el trabajo de *pre* o de *post*. El manejo del *watchdog* lo analizaremos detenidamente en una lección posterior.

Veamos ahora algunas recomendaciones para el uso del TIMER0.

1. Configurando el TIMER0.

Los pasos son los siguientes:

- Asignar el *prescalador*
- Configurar el temporizador colocando los *bits* adecuados en el registro OPCIÓN
- Para inicializar el temporizador, simplemente debemos escribir un número en el registro TIMER0, o borrarlo. Cuando se ha cargado con un número el temporizador cuenta desde ese número, **figura 12.2**.
- Una vez ha empezado a trabajar no es posible detener al TMR0.

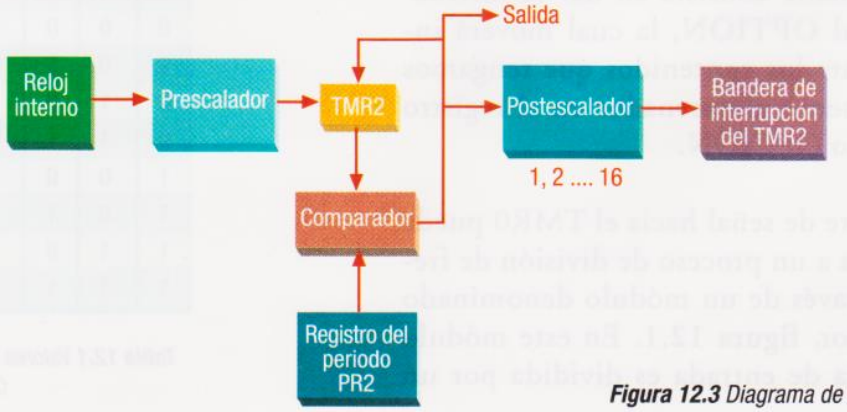


Figura 12.3 *Diagrama de bloques del TMR2*

- El temporizador se mantendrá contando siempre y cuando no se haya borrado o escrito mediante instrucciones del programa o el microcontrolador no se haya inicializado de nuevo.
- El temporizador/contador debe ser recargado después de cada sobreflujo para repetidos intervalos de tiempo. Si esto no se ha hecho, FF resultará en cada ocasión.

2. ¿Cómo nos damos cuenta que el timer0 está haciendo algo?

- Mediante repetidas lecturas en el registro, o chequeando un *bit* de ese registro.
- Mediante interrupción en el momento del sobreflujo de FF a 00. Así es como nos damos cuenta que el temporizador ha terminado de contar. La bandera de interrupción es la salida.

El Timer2

El Timer2 es un temporizador de 8 *bits* que sólo puede configurarse para que su fuente de señal sea el reloj interno, de manera que sólo sirve como temporizador. Cuenta con un módulo prescalador, un módulo postescalador y un registro para el ajuste del periodo. El prescalador divide el tren de pulsos que le llegan por un valor determinado (**1, 4 o 16**).

El diagrama de bloques del TMR2 es el que se aprecia en la **figura 12.3**. Como podemos observar, el TMR2 se incrementa por el reloj interno (fosc/4) desde 0x00 hasta el número establecido por el registro del periodo (PR2). Si en ese momento ocurre otro incremento, la cuenta pasa de nuevo a 0x00.

El registro del periodo se denomina PR2, el cual es de 8 *bits* y sobre él podemos hacer operaciones de lectura o escritura, de manera que podemos modificar el momento del desborde.

En el registro PR2 configuramos un número determinado el cual es vigilado permanentemente por un módulo de comparación, y cada vez que el valor contenido en el TMR2 concuerda con el valor contenido en el registro del periodo, se origina un pulso hacia el postescalador.

En él se ejecuta una nueva división de la salida del comparador y puede activar la bandera de interrupción dentro del registro PIR1 (recordemos de la lección 11, que el registro PIR es propio de cada tipo de microcontrolador según las interrupciones que posea internamente, razón por la cual, el lector debe ubicar en cada hoja de datos, dónde se encuentra la bandera del TMR2 dependiendo del microcontrolador).

De la misma manera, cuando el valor del TMR2 es igual al registro PR2, la salida del comparador inicializa de nuevo el TMR2, lo cual es muy utilizado para la generación de PWM cuando se usa el módulo CCP (lo veremos más adelante), al igual que en aplicaciones seriales (SSP).

Las posibles salidas del TRM2 son la lectura que se puede hacer en cualquier momento del registro TMR2 o la bandera de interrupción originada por el TMR2, la cual se genera cada vez que se igualan PR2 y TMR2.

El TMR2 puede activarse o desactivarse en cualquier momento, mediante un *bit* de control ubicado

Registro: T2CON. CONTROL Timer2

-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit7							*bit0*

***bit* 7:** Se lee desde 0
***bit* 6-3:** TOUTPS3 - TOUTPS0 o *bits* de selección de la salida del postescalador del Timer2
0000 = 1-1 postescala
0001 = 1-2 postescala
•
•
•
1111 = 1-16 postescala
***bit* 2:** TMR2ON o *bit* de activación del Timer2
1: Timer2 está encendido
0: Timer2 está apagado
***bit* 1-0:** T2CKPS1 - T2CKPS0 o *bits* de selección del prescalador del Timer2
00 = prescalador está en 1
01 = prescalador está en 4
1x = prescalador está en 16

Figura 12.4 *RegistroT2CON para el control del TIMER2*

Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
PIR1	TMR2IF							
PIE1	TMR2IE							
TMR2	Registro TMR2							
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
PR2	Registro de periodo del Timer 2							

Figura 12.5 *Registros asociados con el manejo del TMR2*

en el registro T2CON. Los valores presentes en el pre y postescalador se borran por tres razones fundamentales: la escritura sobre TMR2, una escritura sobre T2CON, o por un estado de *reset*, que cuando ocurre, llena a PR2 con unos.

En la **figura 12.4** vemos el registro de control del TMR2; allí podemos ver el *bit* 2 denominado TMR2ON con el cual activamos o desactivamos el funcionamiento del temporizador, los *bits* 3 al 6 que configuran el valor que va a tener el postescalador y los *bits* 0 y 1 los cuales establecerán el valor del prescalador. Veamos los pasos para configurar el modo de operación de este temporizador; en la **figura 12.5** apreciamos una tabla con todos los registros asociados al manejo del TMR2.

En el registro T2CON debemos seleccionar el valor del pre y postescalador y la activación del TMR2. Para habilitar la interrupción de este módulo debemos activar el *bit* TMR2IE del registro PIE1; la ubicación de este *bit* dentro del registro no siempre es la misma.

Depende de la referencia del microcontrolador que estemos usando, por eso es conveniente verificarlo en las hojas de datos.

Del registro de control de interrupciones INTCON debemos activar los *bits* 6 y 7 que corresponden al habilitador global (GIE) y al habilitador de interrupciones de otros periféricos (PEIE). La bandera indicadora de la aparición de esta interrupción se encuentra dentro del registro PIR1 y se denomina TMR2IF; cuya ubicación también depende del microcontrolador. En la **figura 12.6** observamos un código para inicializar el TMR2.

El timer1

El temporizador **Timer1** es un módulo de 16 *bits* que se divide en dos registros de 8 *bits* (TMR1H y TMR1L), sobre los cuales se pueden ejecutar operaciones de lectura y escritura. Este par de regis-

```
CLRF T2CON          ; DETENGO AL TMR2,
                    ; PRESCALADOR = 1:1,
                    ; POSTESCALADOR = 1:1
CLRF TMR2           ; BORRO REGISTRO TMR2
CLRF INTCON         ; DESHABILITO
                    ; INTERRUPCIONES
BSF STATUS, RP0     ; SALTO AL BANCO 1
CLRF PIE1           ; DESHABILITO INTERRUP-
                    ; CIONES DE MODULOS
                    ; PERIFERICOS.
BCF STATUS, RP0     ; REGRESO AL BANCO 0
CLRF PIR1           ; BORRO BANDERAS DE INT.
                    ; PERIFERICAS.
MOVLW 0x72          ; POSTESCALADOR= 1:15,
                    ; PRESCALADOR  = 1:16
MOVWF T2CON         ; TIMER2 APAGADO
BSF T2CON, TMR2ON   ; TIMER2 EMPIEZA A
                    ; INCREMENTARSE
ESPERO_DESBORDE
BTFSS PIR1, TMR2IF  ; HA OCURRIDO UNA INTE-
                    ; RRUPCIÓN POR TMR2?
GOTO ESPERO_DESBORDE ;NO, CONTINUO CON
                    ; EL CICLO.
                    ; SE HA DESBORDADO EL
                    ; TIMER?
BCF PIR1, TMR2IF    ; SI, BORRO LA BANDERA
                    ; Y CONTINÚO.
```

Figura 12.6 *Código para el manejo del TIMER2*

Figura 12.7 Modos de funcionamiento del TIMER1

tros se incrementan desde 0000H hasta FFFFH y al pasar por este último valor, regresan a 0000H. Al igual que el Timer0, el Timer1 puede generar interrupción por desborde, para lo cual requiere que un *bit* de habilitación, presente en el registro PIE, se encuentre activado.

El Timer1 puede ser empleado de diferentes maneras. En primer lugar, lo podemos utilizar de la manera típica para leer los contenidos del registro TIMER1, en cualquier momento del programa, **figura 12.7a.**

La segunda posibilidad que se tiene es la de capturar (grabar) los contenidos del registro TMR1 para ser recuperados en el momento en que un flanco sea leído externamente por el microcontrolador, **figura 12.7b**. Los pulsos que entran al TMR1 incrementan el contador.

Cuando un flanco llega por el pin CCP, los contenidos son capturados y se activa una bandera de interrupción. El valor capturado se encuentra ahora disponible para ser leído.

Los contenidos del TMR1 pueden compararse continuamente con los contenidos de un registro especial de comparación, el cual ha sido cargado previamente con algún valor. Cuando los contenidos del TIMER1 se incrementan y llegan a un valor igual al valor almacenado en dicho registro de comparación, se produce una salida, que puede ser un cambio de nivel en el pin de salida del módulo CCP, una interrupción o ambos, **figura 12.7c.**

La utilización del módulo CCP permite leer el TIMER1 y almacenar el resultado para utilizarlo posteriormente cuando se detecta un flanco de entrada externo, todo esto mientras el microcontrolador está ocupado haciendo otras tareas.

Con el módulo CCP también es posible manejar un pin de salida cuando los contenidos del TIMER1 se incrementan hasta un valor determinado, de manera totalmente independiente de otras tareas que el microcontrolador se encuentre ejecutando.

Descripción del TIMER1

En la **figura 12.8** apreciamos un diagrama general de la estructura del TMR1. Podemos notar que la fuente de señal para el TIMER1 puede ser una señal externa para aplicaciones de cuenta, o los pulsos internos de reloj para apli-

Figura 12.8 Diagrama de bloques del TIMER1

Registro: T1CON. Control del Timer1

-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							*bit* 0

***bit* 7-6** Deshabilitado. Se lee desde 0
***bit* 6-3:** T1CKPS1 - T1CKPS0 o *bits* de selección del valor del prescalador del Timer1
11 = 1- 8 valor del prescalador
10 = 1- 4 valor del prescalador
01 = 1- 2 valor del prescalador
00 = 1- 1 valor del prescalador
***bit* 3:** T1OSCEN o *bit* de habilitación del oscilador del Timer1
1: Oscilador está encendido
0: oscilador está apagado
***bit* 2:** $\overline{\text{T1SYNC}}$ o *bit* de selección de la entrada externa de sincronización al Timer1
Cuando TMR1CS = 1
1: la entrada externa no se sincroniza
0: la entrada externa se sincroniza
Cuando TMR1CS = 0
Ésta es ignorada
***bit* 1:** TMR1CS o *bits* de selección de la fuente para Timer1
1 = entrada externa por el pin T1OSO/T1CKI (flanco de subida)
0 = reloj interno (FOSC/4)
***bit* 0:** TMR1ON o *bit* de habilitación del Timer1
1: habilita Timer1
0: detiene Timer1

Figura 12.9 *Registro de control del TIMER1 (T1CON)*

caciones de temporización, la entrada ingresa a un módulo prescalador que divide los pulsos de entrada por 1, 2, 4 o 8.

La selección de la fuente de señal para el TIMER1 se hace mediante un *bit* específico del registro T1CON, **figura 12.9**. Se trata del *bit* TMR1CS con el cual se establece si la entrada va a ser externa o van a ser los ciclos de reloj internos. De igual manera, a través del *bit* 0 denominado TMR1ON de este registro, podemos activar o desactivar el TMR1 en cualquier momento del programa y mediante los *bits* 4 y 5 (T1CKPS1 y T1CKPS0) se establecen los valores que tendrá el prescalador para dividir la señal de entrada.

Cuando el TMR1 se desborda, se activa un *bit* de bandera del registro PIR1 y se genera una interrupción, si ésta última se ha habilitado previamente mediante el *bit* correspondiente dentro del registro PIE1. Los pulsos externos deben entrar por un pin específico cuyo nombre dentro del encapsulado es T1CKI, **figura 12.10**.

Los pulsos externos que entran al TMR1 pueden ser de dos tipos. El primero, es por pulsos que entran por T1CKI, así el TMR1 se incrementa en cada flanco de subida que se presente en este pin (dividido por el valor especificado en el prescalador). Este modo de operación se habilita con el *bit* TMR1CS perteneciente al registro T1CON. Cuando se desea leer pulsos por el pin T1CKI (*bit* 0 del puerto C), este pin debe estar previamente configurado como entrada.

El segundo tipo de entrada al TIMER1 es a través de un circuito oscilador conectado a los pines T1OSCI y T1OSCO, el cual puede ser hasta de 200 KHz, **figura 12.11**. La aplicación clásica de esta conexión que se encuentra en gran cantidad de notas de aplicación, es la creación de un reloj de tiempo real utilizando este temporizador y un cristal de 32.768 KHz.

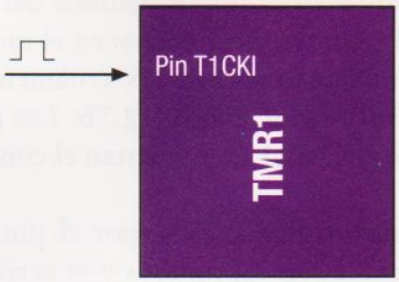


Figura 12.10 *Entrada del TMR1*

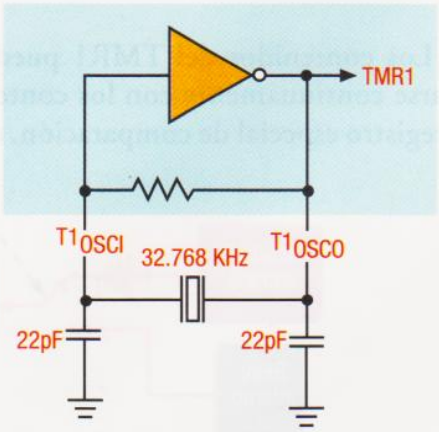


Figura 12.11 *Conexión del TMR1 con un circuito oscilador para la creación de un reloj de tiempo real*

LECCIÓN 13

MÓDULOS CCP (CAPTURA/COMPARACIÓN/PWM)

En la lección anterior conocimos los módulos temporizadores de los microcontroladores PIC que son la base para trabajar con los módulos CCP. Ahora estudiaremos el manejo y funcionamiento de estos últimos, los cuales pueden ser de gran utilidad en muchas aplicaciones en electrónica. Hablamos en plural sobre estos módulos debido a que en un microcontrolador puede existir más de un módulo CCP.

El modo de comparación permite manejar de manera muy efectiva la salida sobre un pin determinado del microcontrolador y el modo de captura permite hacer mediciones muy precisas de los instantes en que se producen algunos flancos de entrada. Ambos modos se han diseñado para trabajar con el **Timer1**; si un microcontrolador posee varios módulos CCP funcionando en modo de captura o comparación, ellos deberán compartir el mismo temporizador. El modo de PWM (modulación por ancho de pulso) es muy utilizado para controlar motores, lámparas u otras cargas resistivas y está diseñado para el **Timer2**, de manera que ningún otro temporizador puede ejecutar esta función, ver **Tabla 13.1**. Cada módulo CCP contiene un registro de 16 *bits* el cual puede trabajar como registro de captura de 16 *bits*, como un registro de comparación de 16 *bits* o como un registro para generar PWM en 10 *bits*. Todos los módulos CCP trabajan prácticamente de la misma manera.

Registro de control: CCPxCON

No implementados	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
*bit*7						*bit*0

***bit* 7-6:** Sin implementar. Se leen como 0
***bit* 5-4:** **DCxB1:DCxB0.** Ciclo útil del PWM
***bit* 3-0:** **CCPxM3:CCPxM0**. Bit de selección del modo CCPx
0000: Captura/compara/apaga PWM (reinicia el módulo CCPx)
0100: Modo de captura, cada flanco de bajada
0101: Modo de captura, cada flanco de subida
0110: Modo de captura de cada 4^0 flanco de subida
0111: Modo de captura de cada 16^0 flanco de subida
1000: Modo de comparación
Inicia pin CCP bajo, cuando la comparación indica igualdad, pone el pin CCP en alto (se activa el *bit* CCPIF)
1001: Modo de comparación
Inicia pin CCP alto, cuando la comparación indica igualdad, pone el pin CCP en bajo se activa el *bit* CCPIF)
1010: Modo de comparación
Genera interrupción por software, cuando la comparación indica igualdad (se activa el *bit* CCPIF, el pin CCP no se afecta)
1011: Modo de comparación
Disparador de eventos especiales (se activa el *bit* CCPIF)
11xx: Modo PWM

Figura 13.1 *Registro de control de los módulos CCP*

Cada módulo CCP posee tres registros para los cuales utilizaremos nombres genéricos, como se muestra en la **Tabla 13.2**. En ella vemos que, cuando dentro de cada registro colocamos una **x**, quiere decir que hablamos indistintamente del módulo 1 o del módulo 2. En la **Tabla 13.3** se muestran las interacciones entre los módulos CCP; en esta tabla, **CCPx** es un módulo CCP y **CCPy** es otro módulo CCP. En la **figura 13.1** tenemos el registro de control de estos módulos CCP con la función de cada uno de sus *bits* constitutivos.

Modo de comparación

El **Timer1** junto con el CCP, pueden controlar de manera muy precisa la salida de un pin del microcontrolador, independiente de las tareas que se encuentre ejecutando la CPU. Algunos microcontroladores como el 16C62A, el 16C64A y el 16C72 solo cuentan con un módulo CCP, ellos solo pueden controlar los tiempos de salida del pin RC2/CCP1, mientras que los microcontroladores que poseen dos módulos CCP pueden controlar también el pin RC1/CCP2. El nombre que reciben estos pines nos indica la función alterna que cumplen, ésta es la de pines de propósito general del puerto C. Recordemos de la lección anterior que el **Timer1** incluye un prescalador que divide la frecuencia interna del reloj por 1, 2, 4 o 8; si seleccionamos la división por 1 obtendremos la resolución más fina para ajustar el tiempo de salida de un flanco de salida. Esto significa que, con un cristal de 4MHz, el tiempo de los flancos de un pulso puede controlarse con una resolución de 1us.

Modo CCP	Función
Captura	Timer1
Comparación	Timer1
PWM	Timer2

Tabla 13.1 *Fuente de cada modo de operación*

Nombre genérico	CCP1	CCP2	Comentario
CCPxCON	CCP1CON	CCP2CON	Control de registro CCP
CCPRxH	CCPR1H	CCPR2H	*Byte* CCP alto
CCPRxL	CCPR1L	CCPR2L	*Byte* CCP bajo
CCPx	CCP1	CCP2	Pin CCP

Tabla 13.2 *Nomenclatura genérica de los registros CCP*

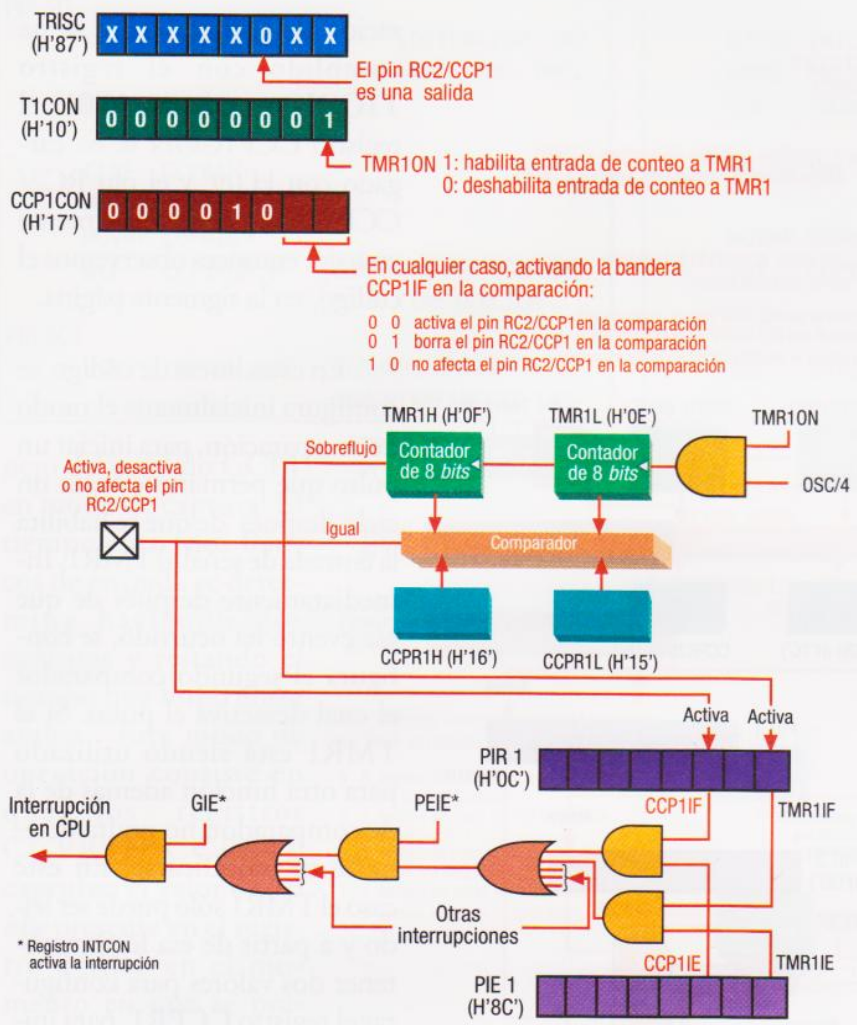


Figura 13.2 *CCP1 en modo de comparación*

En la **figura 13.2** se muestran gráficamente los pasos que se deben seguir para iniciar el módulo de comparación CCP1; allí observamos los registros que se deben configurar: el TRISC, el T1CON y el CCP1CON. En este caso el valor almacenado en los registros CCPR1H y CCPR1L se compara constantemente con los registros del TMR1, los cuales se incrementan progresivamente por su fuente de señal. Cuando ambos son iguales, además de generarse una interrupción, en el pin CCP1 se lleva a cabo una acción determinada por los valores establecidos en los *bits* de control CCP1M3:CCP1M0 (CCP1CON<3:0>); dicha acción puede ser poner en alto, en bajo o permanecer sin cambiar el estado del pin. Cuando se origina un desborde en el **Timer1** o el comparador ha establecido igualdad entre el TMR1 y los registros CCPR1, se origina un estado de interrupción y se activa la bandera correspondiente en el registro **PIR1**. Si se cumple que el habilitador global (**GIE**), el habilitador de interrupciones de periféricos y los *bits* correspondientes del registro **PIE1** se encuentran activados, se generará en la CPU un estado de interrupción. Para microcontroladores que posean dos módulos CCP, el circuito y los registros asociados al CCP2 se muestran el **figura 13.3**, pero su funcionamiento es prácticamente el mismo del circuito anterior; en esta figura podemos apreciar los registros exactos que se deben configurar en este caso, con las respectivas direcciones de cada uno de ellos. Cuando un microcontrolador posee módulo CCP2 debe incluir en su mapa de memoria todos estos registros.

Para microcontroladores PIC que posean dos módulos CCP, si ambos módulos se usan para comparación o captura, deben compartir el TMR1 y en este caso el TMR1 no debe modificarse (escribir sobre él). Pero si el TMR1 sólo se

Modo CCPx	Modo CCPy	Interacción
Captura	Captura	La misma base de tiempo de TMR1
PWM	PWM	Ambos PWM tendrán la misma frecuencia
PWM	PWM	Ninguno
PWM	PWM	Ninguno

Tabla 13.3 *Interacción entre los módulos CCP*

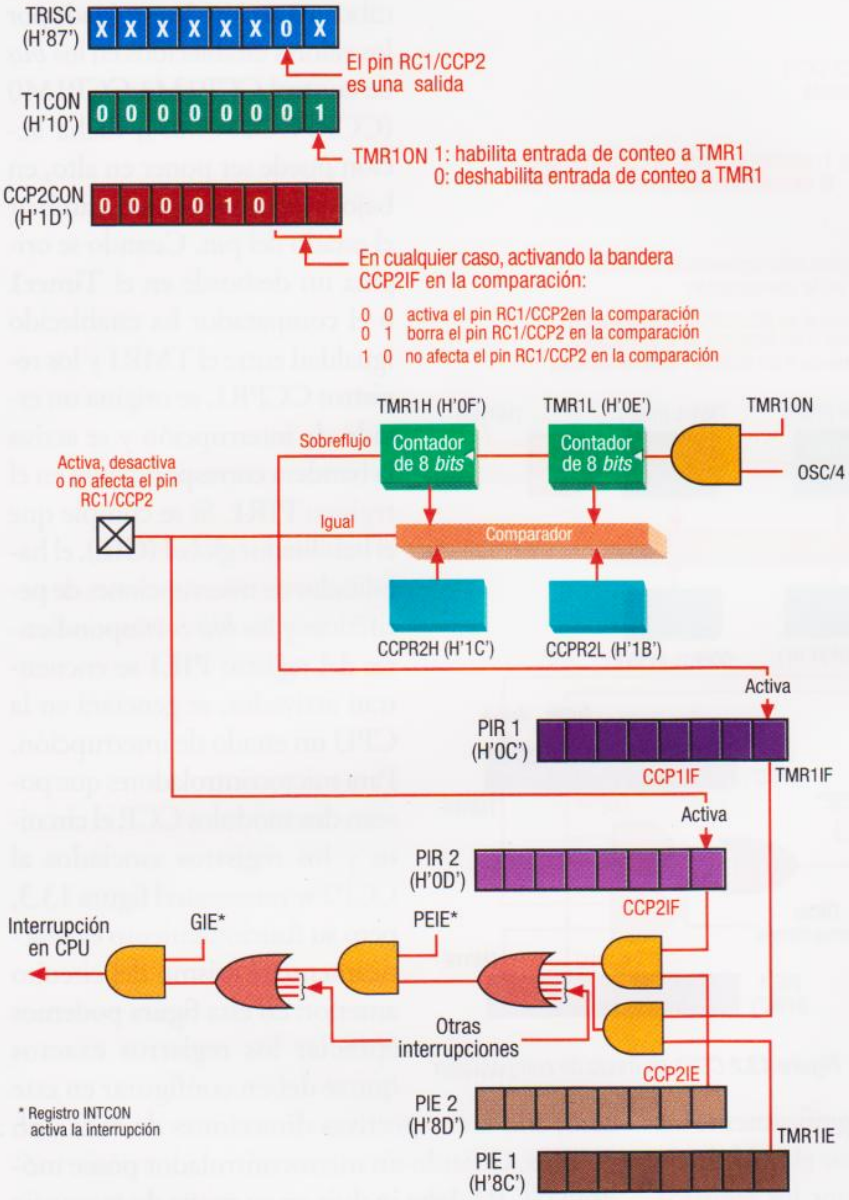


Figura 13.3 *CCP2 en modo comparación*

usa para una función, su uso se puede sintetizar en los siguientes pasos:

1. Detener su reloj (borrando el *bit* TMR1ON del registro T1CON)
2. Borrar el TMR1
3. Configurar el registro CCPR1
4. Iniciar de nuevo la cuenta del TMR1

Veamos el siguiente ejemplo. Asumiendo que el TMR1 no se utiliza para ninguna otra función, originar un pulso positivo de 100 us en el pin RC2/CCP1 en un microcontrolador PIC funcionando con un cristal de 4MHz. Asumamos que la inicialización de la **figura 13.2** se ha cumplido con el registro T1CON cargado con H'01', el registro CCP1CON se ha cargado con H'09' y el pin RC2/CCP1 se ha puesto inicialmente en bajo, entonces observemos el código, en la siguiente página.

En estas líneas de código se configura inicialmente el modo de comparación, para iniciar un pulso que permanece hasta un ciclo después de que se habilita la entrada de señal al TMR1. Inmediatamente después de que ese evento ha ocurrido, se configura el segundo comparador el cual desactiva el pulso. Si el TMR1 está siendo utilizado para otra función además de la de comparador, no podrá detenerse ni modificarse. En este caso el TMR1 sólo puede ser leído y a partir de esa lectura obtener dos valores para configurar el registro **CCPR1**, para iniciar y detener el pulso.

Modo de captura

La combinación del **Timer1** con el CCP1 o el CCP2 permite que el PIC se utilice para determinar el tiempo en que ocurre un flanco de entrada. El **Timer1** y su prescalador pueden utilizarse para medir directamente intervalos de alto rango; la mejor resolución en la medida se obtiene omitiendo el uso del prescalador. De esta manera, con un cristal de 4MHz y omitiendo el prescalador, se pueden medir tiempos de aparición de una entrada en el rango de 1us. Es importante tener claro que en el modo de captura, el pin CCPx debe estar configurado como entrada.

En la **figura 13.4** se muestra el circuito y los respectivos registros asociados al uso del ma-

```
PULSO
      BCF    T1CON,TMR1ON        ;DETENCIÓN  DEL
                                 ;RELOJ DEL TMR1
      CLRF   TMR1H
      CLRF   TMR1L
      CLRF   CCPR1H
      MOVLW  H'01'
      MOVWF  CCPR1L
      BCF    CCP1CON,0     ;CONFIGURO PIN RC2/CCP1
                           ;PARA COMPARACIÓN
PULSO1
      BCF    INTCON,GIE    ;DESHABILITAMOS
                           ;MOMENTANEAMENTE LAS
                                 ;INTERRUPCIONES
      BTFSC  INTCON,GIE
      GOTO   PULSO1
      BSF    T1CON,TMR1ON  ;INICIAN LAS SEÑALES
                           ;AL TMR1
      MOVLW  101           ;CONFIGURAR SEGUNDO
                           ;COMPARADOR
      MOVWF  CCPR1L
      BSF    CCP1CON       ;LIMPIAR PIN RC1/CCP1
                           ;EN EL SEGUNDO
                           ;COMPARADOR
      BSF    INTCON,GIE    ;HABILITAR NUEVAMENTE
                           ;LAS INTERRUPCIONES
```

nejo del módulo CCP1 en modo de captura. El tiempo entre dos flancos de entrada se determina haciendo dos capturas y restando el tiempo que hubo entre ambas. Este modo de operación consiste en que los registros CCPR1H:CCPR1L capturan el valor de 16 *bits* presente en el registro TMR1 en el momento en que se presenta un evento en el pin CCP1. Este evento se selecciona con los *bits* de control CCP1M3, CCP1M2, CCP1M1, CC1PM0 (CCP1CON <3:0>), entre las siguientes posibilidades:

- Cualquier flanco de bajada
- Cualquier flanco de subida
- Cada 4º flanco de subida
- Cada 16º flanco de subida

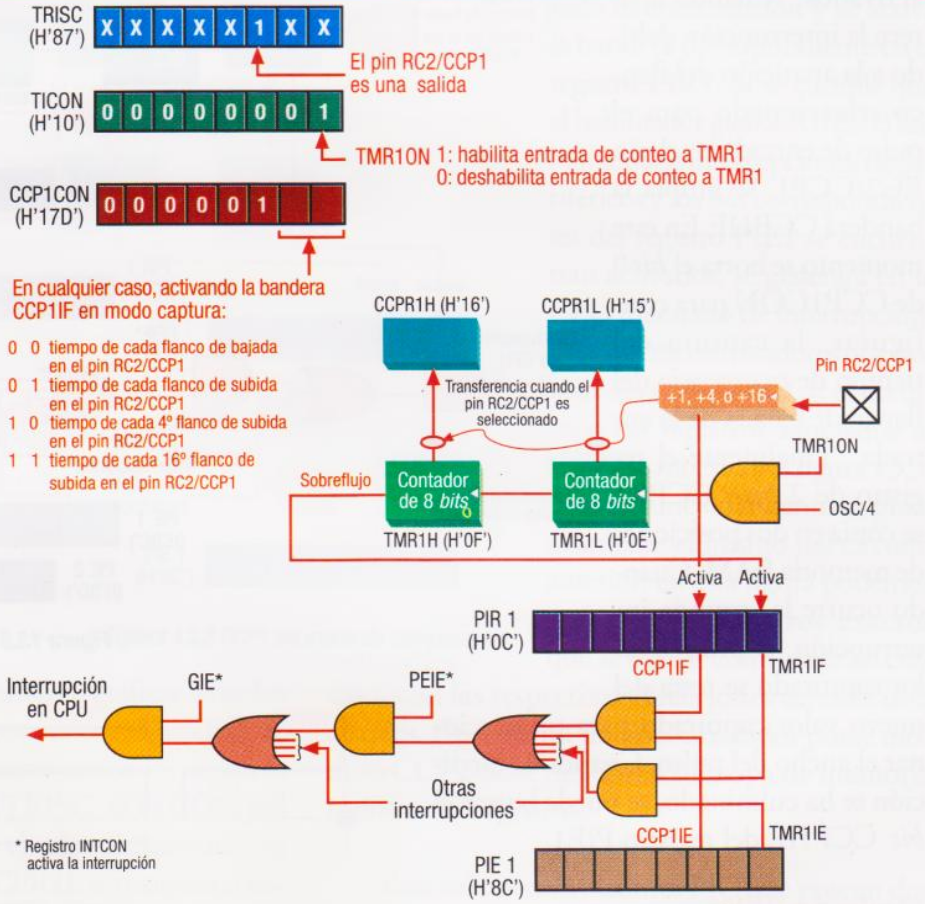


Figura 13.4 *Módulo CCP1 en modo de captura*

En la **figura 13.5** podemos observar el circuito y los registros asociados al funcionamiento y manejo del módulo CCP2, para funcionar en modo de captura; su procedimiento es similar al circuito para manejar el CCP1, pero difiere en algunos registros que se especifican claramente en esta figura. El circuito de la figura 13.4 puede configurarse inicialmente para capturar el tiempo del flanco de subida cargando el registro CCP1CON con H'05'.

Cuando se ha ejecutado una captura, se activa el *bit* CCP1IF, el cual debe limpiarse mediante el programa. Si se ejecuta una nueva captura antes de que el valor del registro CCPR1 se haya leído,

se perderá el valor capturado anteriormente. Luego de borrar la bandera CCP1IF del registro PIR1, se debe activar el *bit* CCP1IE del registro PIE1 para habilitar las interrupciones debidas al módulo CCP1. De igual manera los *bits* GIE y PEIE deben estar activados. Cuando aparece la interrupción debido a la aparición del flanco seleccionado para el pulso de entrada en el pin RC2/CCP1, se limpia la bandera CCP1IF. En este momento se borra el *bit* 0 de CCP1CON para configurar la captura del tiempo de ocurrencia del flanco de bajada de la entrada. Finalmente el registro de 2 *bytes* CCPR1 se copia en dos posiciones de memoria RAM. Cuando ocurre la segunda interrupción, el primer valor capturado se resta del nuevo valor capturado para proporcionar el ancho del pulso. Cuando la medición se ha culminado, se puede borrar el *bit* CCP1IE del registro PIE1.

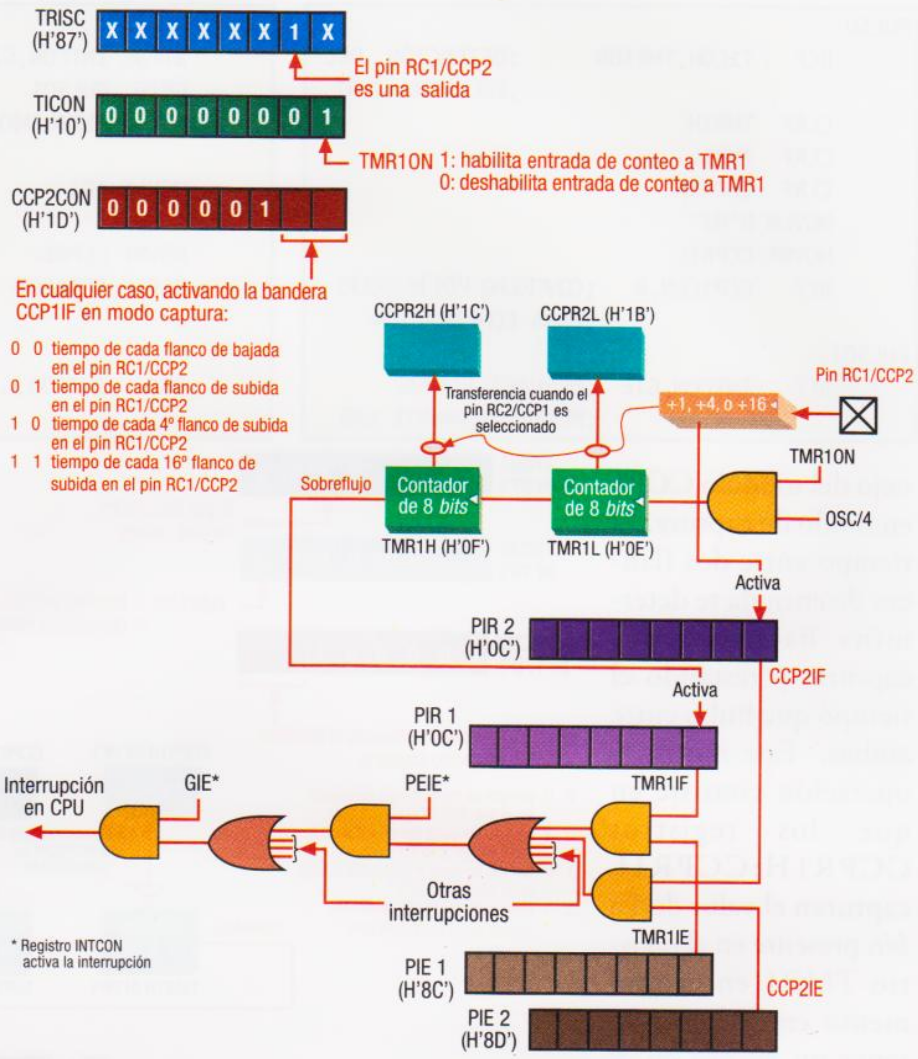


Figura 13.5 *Módulo CCP2 en modo de captura*

Modo de PWM

En el modo de modulación por ancho de pulso (PWM), el pin CCPx produce una salida PWM de 10 *bits*. Debido a que el pin CCPx posee una función alterna como pin I/O, se debe configurar por medio de la instrucción TRIS para que trabaje como salida. En la **figura 13.6** se muestra la salida de modulación por ancho de pulso (PWM) originada desde un microcontrolador PIC; allí se observan los principales elementos que se

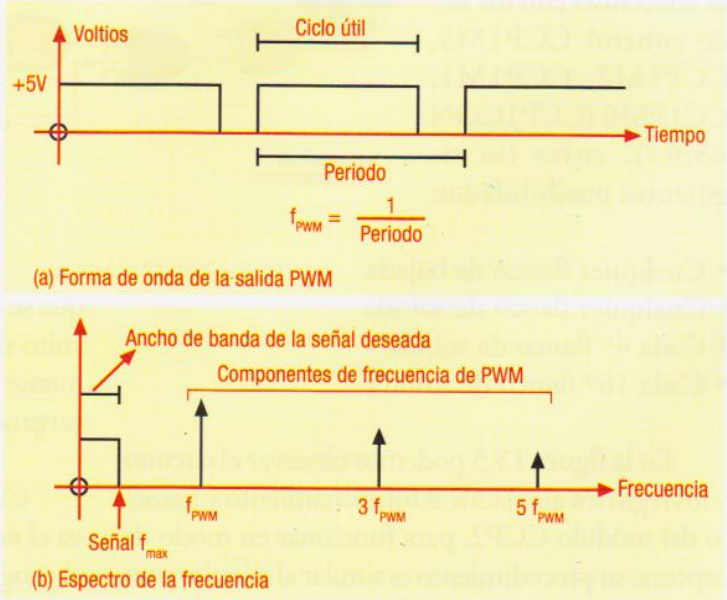


Figura 13.6 *forma de onda de una señal PWM*

distinguen en este tipo de ondas, a saber: el periodo, el ciclo y la frecuencia de la señal de PWM.

En muchas aplicaciones prácticas es necesario un periodo de la señal corto. Por ejemplo, el circuito de la **figura 13.7** utiliza una salida variable de PWM para generar voltajes entre 0 y 5 voltios, utilizando un filtro pasabajo para conformar un valor promedio a partir de la forma de onda. Entre más corto sea el periodo de la PWM, más rápido podrá cambiar el valor promedio. Si los cambios en la señal PWM se caracterizan por una frecuencia máxima, *fmax* como se muestra en la **figura 13.6b**, se tiene:

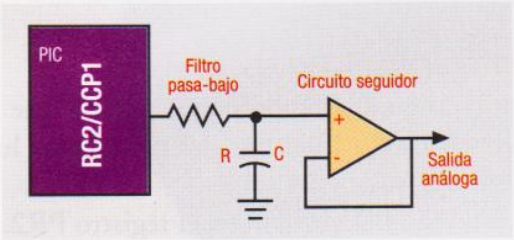


Figura 13.7 *Circuito de conversión D/A*

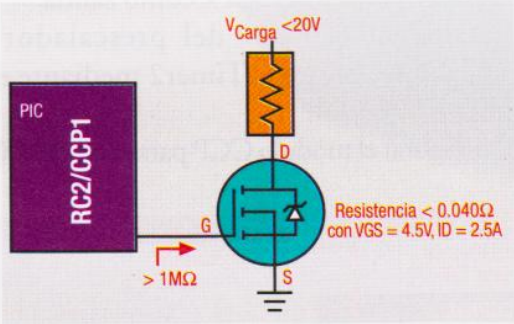


Figura 13.8 *Manejo de una carga conmutada*

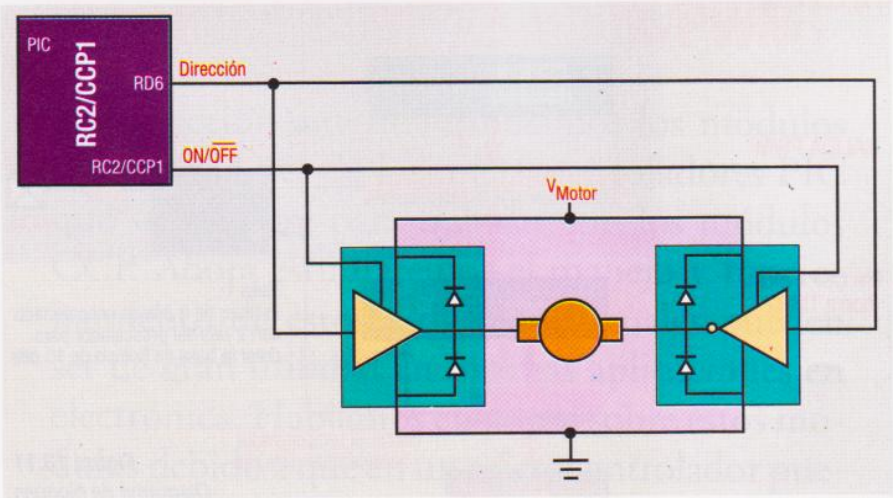


Figura 13.9 *Control de un motor CC con un puente H*

fmax << fcorte del filtro << f pwm

La frecuencia de corte del filtro necesita estar por debajo de la frecuencia de la onda PWM, *fpwm*, para remover su componente fundamental y sus armónicos en 3 fpwm, 5 fpwm, 7 fpwm, etc. y dejar solamente los componentes CC que varíen lentamente. De esta manera, acortar el periodo de la señal de la PWM es la manera más simple de acomodar señales de frecuencia con un filtro que las deje pasar y remueva los efectos de la modulación PWM.

En algunas otras aplicaciones el filtrado es innecesario. En la **figura 13.8** se muestra un circuito que puede usarse para controlar temperatura o la intensidad de una fuente de luz mediante un MOSFET; dispositivo que para estos casos es ideal. Este elemento se caracteriza por poseer una entrada de alta impedancia que lo hace fácil de manejar. Para controlar adecuadamente un motor CC se necesita un número o magnitud y un signo; el signo nos indica la dirección hacia la cual va a girar el motor y la magnitud es la cantidad de energía que se le va a aplicar al movimiento; en el circuito de la **figura 13.9**, el signo se origina por el pin RD6. Cuando dicho *bit* se encuentra en alto y la salida de la PWM activada, la parte izquierda del circuito se activa haciendo que la corriente fluya hacia la derecha. Si el *bit* 6 se pone en bajo y la salida de la PWM se encuentra activada, la corriente fluye de derecha a izquierda en el motor.

El concepto del funcionamiento de la PWM es el que se muestra en la **figura 13.10**. El TMR2 es sometido a dos comparaciones con dos registros distintos, la primera es con el registro del ciclo útil y la segunda es con el registro del periodo. Cuando el TMR2 y el ciclo útil son iguales, la salida del pin CCPx se pone en bajo, pero el TMR2 continúa incrementándose hasta igualar al registro del pe-

riodo; cuando esto ocurre, el pin CCPx se pone en alto y se borra el TMR2 para iniciar de nuevo.

En la **figura 13.11** se muestra un diagrama en bloques del módulo CCP funcionando en modo PWM. El periodo de la PWM se especifica en el registro PR2 y se calcula mediante la siguiente fórmula:

Periodo de la PWM = [(PR2) + 1] • 4 • Tosc • (valor del prescalador del TMR2)

Cuando el TMR2 es igual a PR2, en el siguiente ciclo se presentan los siguientes eventos:

- El TMR2 es borrado
- El pin CCPx se pone en alto
- El ciclo útil de la PWM es almacenado en CCPRxL pasa a CCPRxH

Ciclo útil de la PWM

El ciclo útil de la PWM se especifica en el registro CCPRxL y en los *bits* DCxB1:DCxB0 (CCPxCON<5:4>). Se puede lograr una resolución hasta de 10 *bits* gracias a los 8 de CCPRxL y a los *bits* 4 y 5 de CCPxCON que actúan como los *bits* LSB. Este valor de 10 *bits* en definitiva queda en DCxB9:DCxB0. Para calcular el ciclo útil nos podemos remitir a la siguiente fórmula:

Ciclo útil de la PWM = (valor en los *bits* DCxB9:DCxB0) • Tosc • (valor del prescalador del TMR2)

Los *bits* DCxB9:DCxB0 pueden ser escritos en cualquier momento, pero el valor del ciclo útil no se almacena en CCPRxH hasta después de que los contenidos de PR2 y TMR2 sean iguales, lo cual marca el final del periodo actual. En el modo PWM el registro CCPRxH solo puede ser leído. El registro CCPRxH y un *latch* interno de 2 *bits* configuran el *buffer* del ciclo útil, el cual, al igualar el contenido del TMR2 concatenado con dos *bits* de su prescalador, borran el pin CCPx, estableciendo el final del ciclo útil.

Configuración de la PWM

Para configurar el módulo CCP para que funcione en modo PWM se deben ejecutar los siguientes pasos:

1. Establecer el periodo en el registro PR2.
2. Establecer el ciclo útil en los *bits* DCxB9:DCxB0.
3. Configurar el pin CCPx como salida.
4. Establecer el valor del prescalador del TMR2 y habilitar el Timer2 mediante el registro T2CON.
5. Configurar el módulo CCP para el modo PWM.

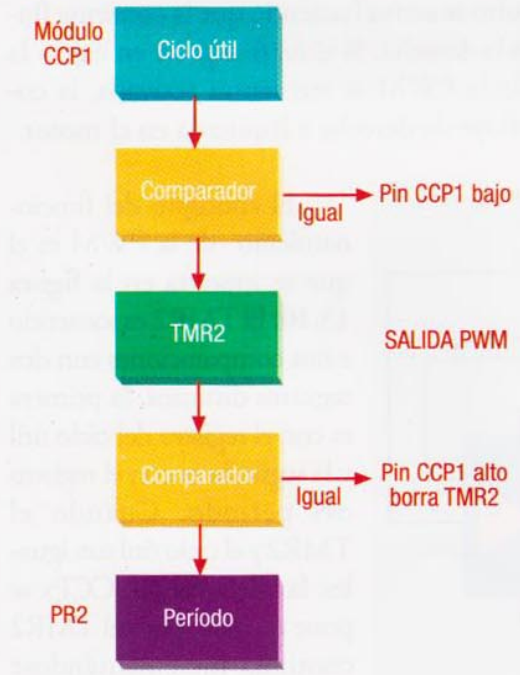


Figura 13.10 Concepto del funcionamiento de la salida PWM

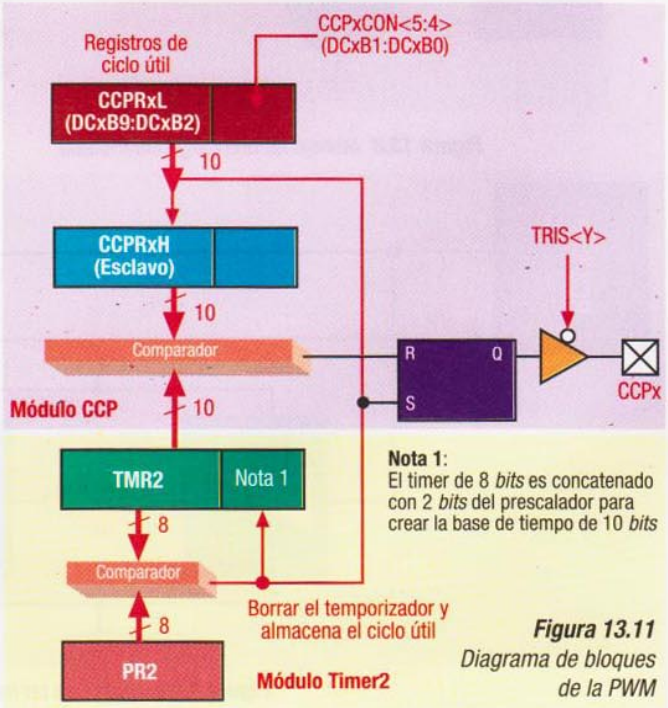
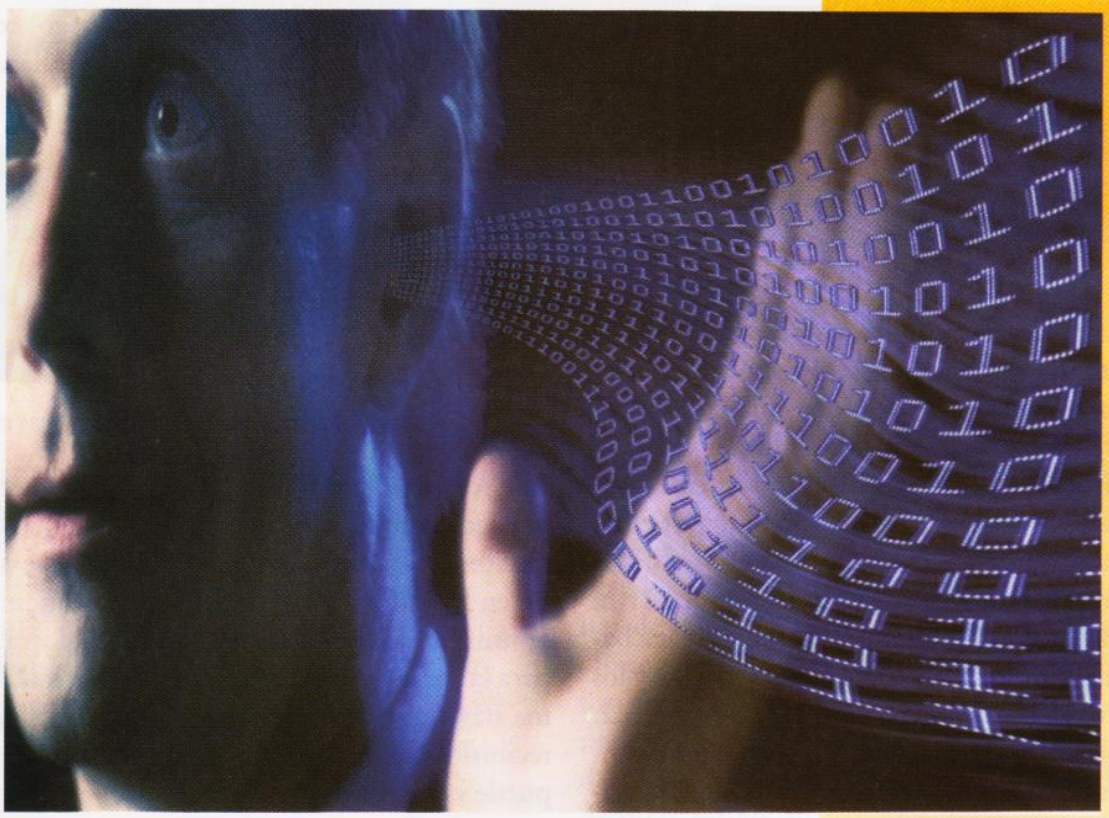


Figura 13.11 Diagrama de bloques de la PWM

LECCIÓN 14



INTRODUCCIÓN A LAS COMUNICACIONES SERIALES

La utilización de las comunicaciones seriales se encuentra por doquier en el mundo digital ya que nos ofrece una solución económica y de alta calidad para una gran variedad de aplicaciones.

En los sistemas con microcontroladores siempre existe la necesidad de conectar elementos internos entre sí como los microcontroladores con las memorias, con conversores A/D, conversores D/A, etc. o entre el microcontrolador y elementos externos al sistema como teclados, displays e interfaces de potencia, entre otros.

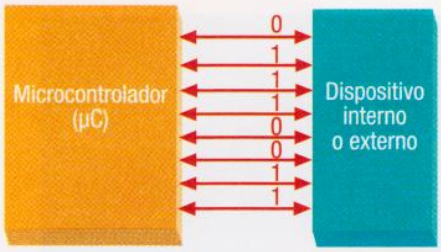


Figura 14.1 *Comunicación en paralelo*

En términos generales, hay dos maneras de interconectar elementos electrónicos digitales: en paralelo y en serie. Se dice que una comunicación es en paralelo cuando se conectan varias líneas entre los dispositivos (internos o externos) y los datos se envían simultáneamente por ellas; en la mayoría de los casos, en valores múltiplos de 8 bits (8, 16, 32, 64, etc.), **figura 14.1**. El otro tipo de comunicación, la serial, se hace básicamente por una sola línea (o muy pocas) y los bits se envían uno detrás de otro, **figura 14.2**. En el primer caso, la comunicación es más rápida y ocupa muchas líneas o pines del microcontrolador y en el segundo, es más lenta y utiliza pocos pines lo que nos permite realizar otras funciones con los pines libres. Actualmente, debido a la alta velocidad que se ha logrado en la operación de estos circuitos, la comunicación serial es la más utilizada ya que conviene más ahorrar pines, que tener en cuenta la velocidad de transmisión que se lograría con la conexión en paralelo.

Las familias de rango alto de los microcontroladores PIC tienen incorporados módulos internos de comunicaciones con sus pines o terminales especializados para esta función y sus correspondientes instrucciones de programación lo que facilita estos procesos. En las familias de bajo nivel (pocos pines), las comunicaciones se hacen por medio de los pines de entrada/salida (I/O) o puertos y los protocolos (reglas de juego) se deben establecer en el programa lo que hace un poco más compleja su elaboración.

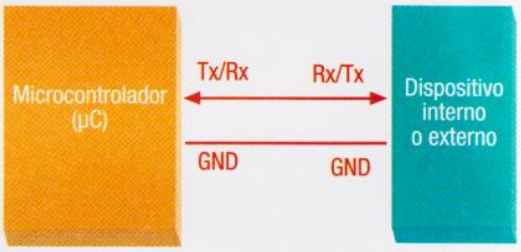


Figura 14.3 *Comunicación unidireccional (half duplex)*

Por otro lado, las comunicaciones seriales pueden ser unidireccionales (half duplex) o bidireccionales (full duplex). En las primeras se transfieren los datos en una sola dirección al tiempo y se utilizan dos cables, uno para la señal y el otro para la masa o tierra, **figura 14.3**. Con este sistema hay que enviar primero los datos y después recibirlos o viceversa. Con las bidireccionales, se puede enviar y recibir información al tiempo y se requieren como mínimo tres cables o pines, uno para enviar, otro para recibir y otro para la masa o tierra, **figura 14.4**. En las comunicaciones seriales hay dos métodos principales: las comunicaciones sincrónicas o sincronizadas y las asincrónicas o no sincronizadas lo cual explicaremos más adelante.

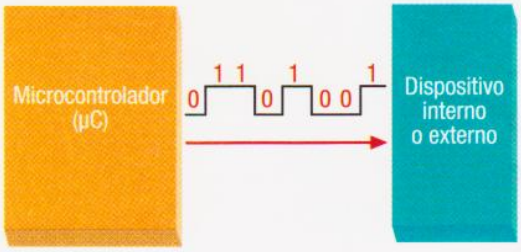


Figura 14.2 *Comunicación en serie o serial*

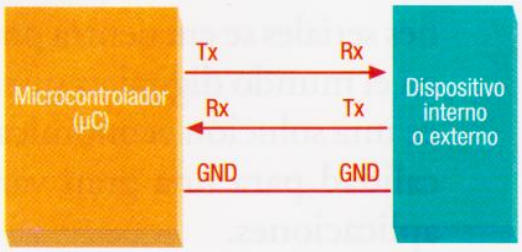


Figura 14.4 *Comunicación bidireccional (full duplex)*

Otro factor que se debe tener en cuenta en las comunicaciones seriales es la velocidad de transmisión de los datos la cual se mide en *bits* por segundo o baudios que es en realidad la velocidad con que cambian los estados de la señal (de 1 a 0 o de 0 a 1). A esta velocidad se le llama generalmente rata de baudios (Baud rate) o simplemente rata. Por lo general, se utilizan ratas con valores normalizados de 2400, 9600 y 19200 baudios. Esto es muy importante ya que en una comunicación serial entre dos o más elementos, se debe conocer la velocidad de transmisión de los datos con el fin de que ésta se logre.

Para que dos equipos se comuniquen satisfactoriamente, ambos deben manejar un mismo conjunto de normas que indiquen los parámetros y la manera correcta de realizar la transmisión y la recepción. Este conjunto de normas es lo que se conoce con el nombre de **protocolo**.

Cuando efectuamos una comunicación serial, los grupos de datos que se van a transmitir se deben descomponer en *bits* los cuales son transmitidos uno por uno desde el emisor hasta el receptor en donde se establece de nuevo el grupo original.

Un carácter lo podemos definir como la unidad mínima de información compuesta de varios *bits*. El emisor puede generar unidades de información de más de 8 *bits*, pero como un microcontrolador PIC tiene una CPU de 8 *bits*, las unidades de información deben dividirse en varios segmentos de 8 *bits* para ser enviados por separado. El dispositivo receptor se encarga de ensamblar de nuevo los segmentos para recuperar el mensaje original. Para propósitos de la transmisión, cada segmento de 8 *bits* se trata como un carácter, sin importar que no sea la unidad final de la información. El ejemplo más clásico de este caso son los conversores A/D de 10, 12 o más *bits*.

Cuando se transmiten caracteres serialmente, los *bits* que conforman cada carácter se envían de manera secuencial, distribuidos en el tiempo. Cuando un dispositivo transmisor envía estos *bits*, el receptor debe estar en capacidad de:

- Determinar el momento exacto en que deben llegar.
- Reconocer cuándo empieza y cuándo termina cada uno de los *bits*.
- Reconocer cuándo empieza y cuándo termina la serie de *bits* que conforman el carácter.

Para facilitar el reconocimiento claro de todos los caracteres, se ha establecido un sistema de sincronización por medio de algunos *bits* de delimitación y separación. Como ya lo mencionamos, existen dos métodos de delimitación que definen los dos tipos básicos de transmisión de datos: la transmisión sincrónica y la transmisión no sincrónica.

Transmisión sincrónica

En este tipo de transmisión se asocia un pulso de reloj con cada *bit* transmitido. En este caso se requieren dos líneas de comunicación (usualmente cables), uno para los *bits* de datos y otro para los pulsos de reloj.

El receptor puede reconocer fácilmente los *bits* de datos debido a que ellos ocurren de manera simultánea con los pulsos de reloj, de esta manera sabremos exactamente cuando leer los *bits* de datos correctos sin importar que la señal de reloj no mantenga una frecuencia estable, **figura 14.5**.

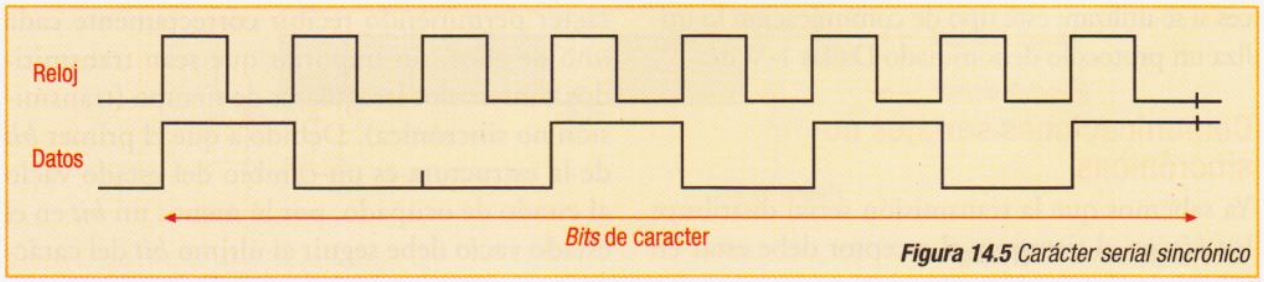


Figura 14.5 *Carácter serial sincrónico*

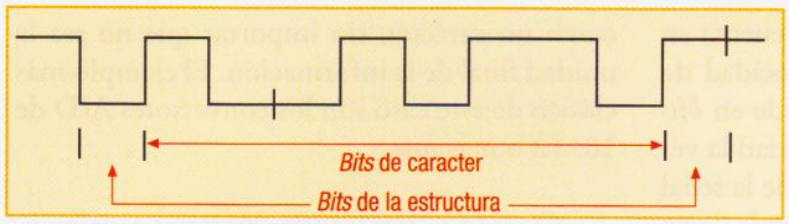


Figura 14.6 *Transmisión no sincrónica*

Cuando un dato se codifica en determinado número de *bits*, para separar los distintos caracteres el receptor necesita contar *bits*, empezando con el primer *bit* recibido. Este tipo de transmisión es adoptado por algunos estándares internacionales como el protocolo SPI, I2C y Microwire.

Transmisión no sincrónica

Este tipo de transmisión asocia unos *bits* especiales a cada carácter conformando una estructura o paquete; incorpora un *bit* justo antes de empezar el carácter y otro *bit* después del carácter. Todos los *bits*, incluyendo los de la estructura, se envían a la misma velocidad (todos los *bits* con un mismo periodo) y cuando llegan al receptor, éste ya sabe que el primer *bit* recibido corresponde al *bit* de estructura y que después de él, en intervalos fijos de tiempo, encontrará los *bits* de datos, **figura 14.6**.

Tanto el emisor como el receptor deben trabajar a la misma velocidad, es decir, a la misma rata de *bits*; como el primer *bit* es quien informa la aparición de un paquete y después de él se localizan los *bits* de datos a una rata constante, el carácter puede ser enviado en cualquier momento. Por ello, este método se conoce como carácter no sincrónico y *bit* sincrónico.

Otro método para la comunicación no sincrónica consiste en transmitir a ratas de *bit* variables. Este método no requiere una longitud fija de los caracteres ni de los *bits* de estructura, aunque a veces sí se utilizan; este tipo de comunicación lo utiliza un protocolo denominado Dallas 1-Wire.

Comunicaciones seriales no sincrónicas

Ya sabemos que la transmisión serial distribuye los *bits* en el tiempo y el receptor debe estar en capacidad de reconocer cuando empieza un *bit* y cuando termina. Si el receptor puede determinar cuando llega el primer *bit* de un carácter y conociendo el hecho de que cada *bit* posee un periodo fijo, el receptor puede anticipar cuando llegarán cada uno de los *bits*. Cuando no existen datos en el medio de transmisión, se dice que se encuentra en estado vacío o desocupado y el receptor se encuentra esperando que llegue un *bit*.

Cuando llega el primer *bit* el receptor lo reconoce y espera un tiempo determinado para leer de nuevo el siguiente *bit*, este primer *bit* forma parte de la estructura y se le llama *bit* de **sincronización**. El receptor repite este proceso de lectura y espera hasta que se hayan leído todos los *bits*.

El transmisor y el receptor poseen cada uno su propio reloj para medir el tiempo de cada *bit*. ¿Es posible que estos dos relojes en un momento dado se desajusten y no queden completamente sincronizados entre sí? En realidad sí, y si ese es el caso, se puede originar un desfase y crear problemas como el del receptor esperando un *bit* en el instante incorrecto.

Este problema se resuelve sincronizando periódicamente, haciendo fija la longitud del carácter y transmitiendo nuevamente el *bit* de sincronización al inicio de cada carácter de tamaño fijo. De esta manera los relojes sólo necesitan estar en fase durante un corto tiempo ya que ellos se ponen en fase nuevamente en cada *bit* de sincronización.

Con un número fijo de *bits* por carácter, el receptor cuenta los *bits* para delimitar cada carácter permitiendo recibir correctamente cada uno de ellos, sin importar que sean transmitidos a intervalos irregulares de tiempo (transmisión no sincrónica). Debido a que el primer *bit* de la estructura es un cambio del estado vacío al estado de ocupado, por lo menos un *bit* en el estado vacío debe seguir al último *bit* del carác-

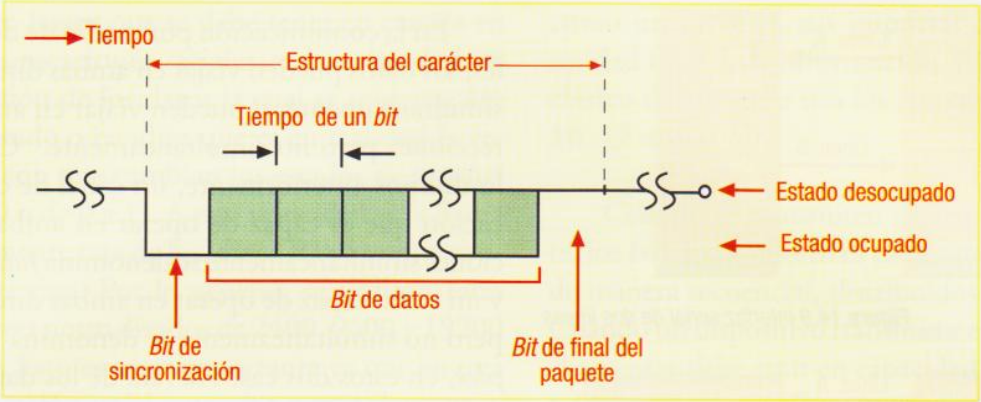


Figura 14.7 *Estructura del carácter*

ter, el cual también corresponde a un *bit* fundamental para conformar la estructura.

Protocolo serial no sincrónico

Este protocolo es utilizado por la mayoría de los módulos especializados para comunicaciones (UART) que existen dentro de los microcontroladores en comunicaciones seriales no sincrónicas. La transmisión se describe en la **figura 14.7**: caracteres transmitidos a intervalos indeterminados pero a una rata fija de *bit* (carácter no sincrónico, *bit* sincrónico).

Según este protocolo, el estado vacío de la transmisión es un 1 lógico, y se le conoce como **estado de marca**. El estado de ocupado es un 0 lógico y se le denomina **estado de espacio**. Al *bit* de sincronización se le llama ***bit* de inicio** y al *bit* de final de la estructura ***bit* de parada**. El *bit* de inicio es una transición desde el estado de marca hacia el estado de espacio y el *bit* de parada es el estado de marca.

Sólo puede existir un *bit* de inicio, pero pueden existir varios de parada. El protocolo especifica que deben ser 1,1/2 o 2 *bits* de parada como mínimo y 5, 6, 7 u 8 *bits* de datos en el carácter. El carácter de datos se transmite empezando por el *bit* menos significativo y a continuación del último *bit* de datos del carácter puede haber (aunque no es completamente necesario) un *bit* especial denominado ***bit* de paridad**.

El *bit* de paridad es un método básico para la detección de errores en la transmisión. La paridad es una característica definida sobre cada carácter de datos. Se determina por el número de unos (1) lógicos en los datos. El primer paso es decidir si se va a trabajar con paridad par o paridad impar. Para paridad impar el número de unos en el carácter de datos debe ser impar. Esto es, se cuenta el número de unos presentes en el carácter de datos y si la cuenta es impar entonces el *bit* de paridad se pone en cero. Si el número de unos en el carácter de datos es par, entonces el *bit* de paridad se pone en uno para hacer el número de unos de nuevo impar, **figura 14.8**. Igual metodología se aplica para el caso contrario, es decir para paridad par.

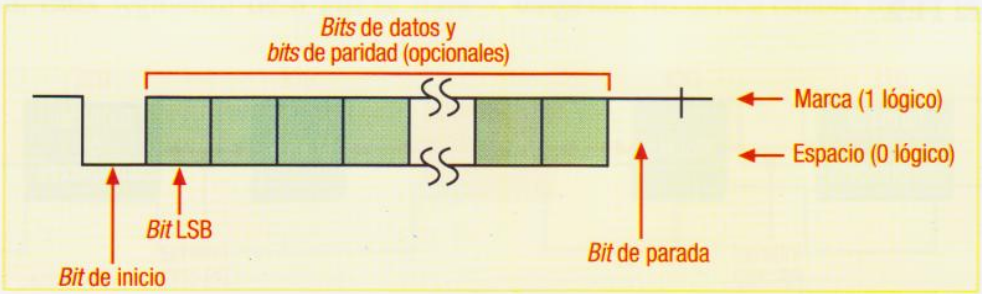


Figura 14.8 *Formato del carácter*

Figura 14.9 *Interfaz serial de dos líneas*

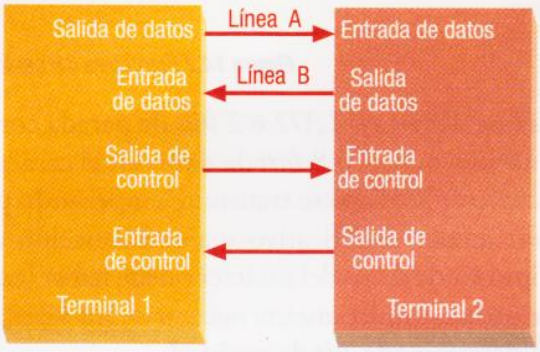


Figura 14.10 *Interfaz serial de dos hilos con líneas adicionales de control*

El canal de comunicación

Muchas comunicaciones se desarrollan en dos vías y requieren de un transmisor y de un receptor al final de las dos vías del canal de comunicaciones. En muchos casos se acostumbra dar el nombre de **equipo terminal** a cada uno de los equipos de comunicación y a las líneas de transmisión se les denomina **ruta de los datos**.

La comunicación en dos vías requiere de dos líneas, una para cada dirección de viaje de los datos. El transmisor de un terminal se conecta por medio de la línea A al receptor del segundo terminal y de la misma manera el transmisor del segundo terminal se conecta al receptor del primero, **figura 14.9**.

En la comunicación por medio de dos canales, los datos pueden viajar en ambas direcciones simultáneamente, o pueden viajar en ambas direcciones pero no simultáneamente. Como ya lo dijimos anteriormente, un canal de comunicación que es capaz de operar en ambas direcciones simultáneamente se denomina *full-duplex;* y un canal capaz de operar en ambas direcciones pero no simultáneamente se denomina *half-duplex,* en estos dos casos la rata de los datos debe ser la misma en ambas direcciones. Un canal de una sola línea capaz de operar en una sola dirección se denomina canal *simplex.*

Para un canal *half-duplex* se debe hacer un sistema de control para prevenir que ambos terminales intenten transmitir al mismo tiempo o que un terminal envíe cuando el otro no está listo para transmitir. Este flujo de control requiere de otras líneas adicionales, utilizadas para controlar de manera adecuada el flujo de los caracteres transmitidos por las líneas de datos, **figura 14.10**.

El estándar RS-232

Una aplicación real de la comunicación serial no sincrónica es la que se ha definido en el estándar internacional EIA RS-232. Se trata de una norma que define las características físicas que debe tener el canal y las funciones de las líneas de control y de datos. No se enfoca en el formato de los datos, de manera que cada usuario es libre de construir sus propios formatos.

Dentro de esta norma, los equipos terminales toman unos nombres específicos dependiendo de su labor dentro del canal de comunicación, ellos pueden ser: Equipo Terminal de Da-

Figura 14.11 *Canal de comunicación serial*

Líneas RS-232 (conector de 25 pines)

PIN		Nombre	Dirección		Función
			DTE	DCE	
1	PG	Tierra de protección	--	--	Tierra de seguridad
2	TD	Datos TX	Salida	Entrada	Datos de salida
3	RD	Datos RX	Entrada	Salida	Datos de entrada
4	RTS	Petición para enviar	Salida	Entrada	DTE quiere enviar
5	CTS	Listo para enviar	Entrada	Salida	OK para que DTE envíe
6	DSR	Datos listos	Entrada	Salida	DCE listo para comunicarse
7	--	Común	--	--	Tierra
8	DCD	Detección de portadora	Entrada	Salida	Enlace de comunicación en proceso
20	DTR	Equipo terminal listo	Salida	Entrada	DTE listo para comunicarse
22	RJ	Indicador de llamada	Entrada	Salida	Anuncia en llamado entrante
23	DSRD	Rata de la señal de datos	Cualquiera	Cualquiera	Indicador de la rata de datos

Tabla 14.1 *Líneas RS-232 en un conector DB-25*

Conector DB-9 IBM-PC

Línea	Conector RS-232	Conector RS-232
	25 pines	9 pines
TD	2	3
RD	3	2
RTS	4	7
CTS	5	8
DSR	6	6
Común	7	5
DCD	8	1
DTR	20	4
RI	22	9

Tabla 14.2 *Conector RS-232 de 9 pines (DB-9)*

tos (DTE) o Equipo de Comunicación de Datos (DCE). Inicialmente podemos entender un DTE como una computadora y un DCE como un módem, **figura 14.11**.

El estándar define 25 líneas con sus correspondientes números de pines dentro de un conector. De las 25 líneas 11 tienen que ver con la comunicación no sincrónica y se pueden apreciar en las **Tablas 14.1** y **14.2**, donde se muestra la distribución para conectores de 25 (DB-25) y 9 pines (DB-9). De la misma manera, según el estándar, el equipo DTE debe tener un conector tipo macho y el DCE un conector tipo hembra.

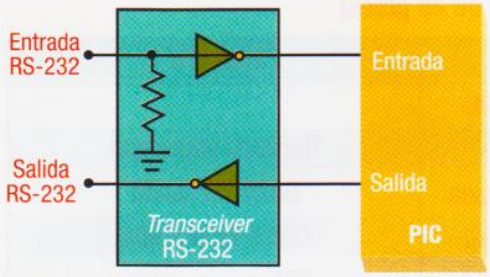
Estas líneas se puede clasificar en dos grupos: las de funciones de datos y las de funciones de control. Solo hay dos líneas para funciones de datos, por la línea 2 se transmiten los datos y por la línea 3 se reciben. Las demás líneas se utilizan para efectos de control del flujo de los datos y el nombre de cada una de ellas refleja su función desde el punto de vista del equipo DTE.

La interfaz RS-232 ha llegado a estandarizar la comunicación entre dos terminales de datos por medio de un módem, un enlace de comunicación y otro módem, **figura 14.12**.

Características eléctricas

El estándar RS-232 se aplica a ratas de datos de hasta 20.000 *bits* por segundo y hasta 15,24

Figura 14.12 *Comunicación serial por medio de un módem*



Lógica	RS-232	PIC
Marca (1)	-3 a 12V	5V
Espacio (0)	+3 a +12V	0V

Figura 14.13 *Interfaz RS-232 con un microcontrolador*

metros de longitud del cable, aunque esta longitud se puede extender por medio de unos dispositivos repetidores de señal que corrigen los niveles de voltaje.

Como estamos trabajando con comunicaciones binarias, solo se han definido dos posibles estados para la señal: marca y espacio. Una marca es un uno lógico y su señal de voltaje, para que sea comprendida correctamente, debe estar en el rango de –3 hasta 15 voltios.

Un espacio es un cero lógico y su voltaje debe estar entre +3 y +15V para que el receptor lo pueda leer adecuadamente. Si una señal se encuentra en el rango comprendido entre –3 y +3V se entiende como un estado indeterminado.

Cuando utilizamos comunicaciones seriales externas en un microcontrolador, las salidas digitales de 0 y 5 voltios deben acondicionarse para llegar a los niveles RS-232. Este problema se puede resolver fácilmente por medio de una configuración de transistores o por medio de un circuito integrado especial, al cual se le entregan los niveles TTL y arroja los niveles RS-232, **figura 14.13**. De este tipo de integrados existe una gran variedad y muchas empresas fabricantes de circuitos los han desarrollado satisfactoriamente.

El código ASCII

Cuando queremos transmitir caracteres que los humanos podamos leer y entender, debemos codificarlos por medio de una combinación de *bits*. El código de texto de más aceptación se llama el código ASCII.

ASCII es un conjunto de caracteres codificados en 7 *bits* que cuenta con 95 caracteres imprimibles y 33 no imprimibles. Los primeros incluyen letras minúsculas, mayúsculas, los números del 0 al 9, caracteres de puntuación y otros.

Los segundos se denominan caracteres de control y sirven para manejar la comunicación. Los caracteres imprimibles también se conocen como caracteres gráficos o leíbles por humanos y son codificados desde $20 hasta $7E como se muestra en la **Tabla 14.3**.

Nibble bajo

Nibble alto	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Tabla 14.3 *Código ASCII*

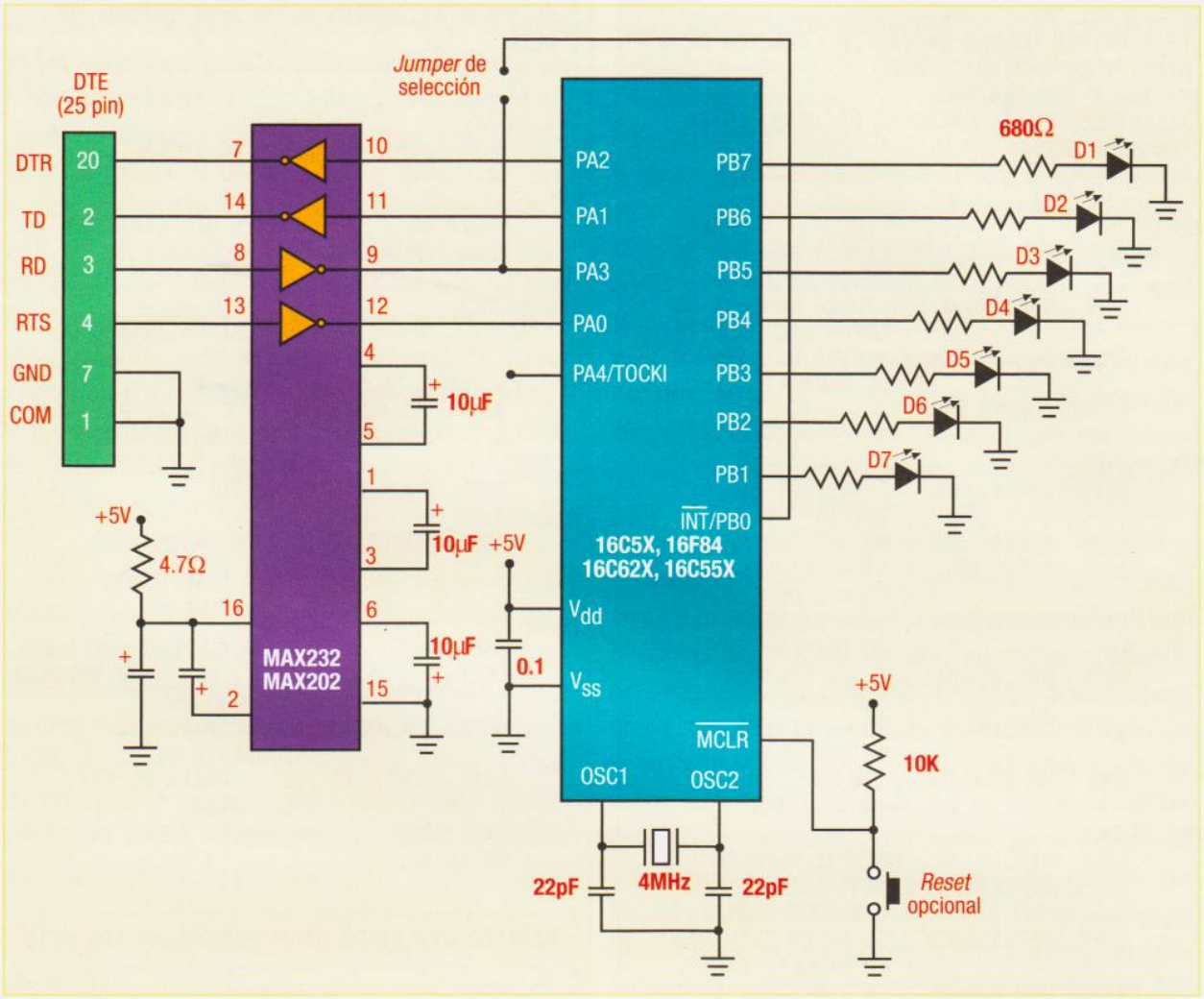


Figura 14.14 *Esquema del circuito para comunicación serial*

La mejor manera de aprender el manejo y el establecimiento de comunicaciones seriales con microcontroladores es a través de algunos ejemplos básicos. Ahora entraremos a analizar dos casos típicos sobre microcontroladores PIC de 18 pines: las comunicaciones *Half-Duplex* y *Full-Duplex*. Daremos algunas rutinas básicas genéricas que no están ligadas a ningún microcontrolador en particular y que pueden ejecutarse sobre la configuración de circuitos mostrada en la **figura 14.14**.

Comunicación no sincrónica en microcontroladores

Como ya lo mencionamos, los microcontroladores más pequeños no cuentan con módulos especializados de comunicación (UART). Sin embargo, esas funciones se pueden desarrollar con algunas líneas de código. En muchas ocasiones, esta opción representa una efectiva y económica solución, y por lo tanto debe ser considerada. El circuito de la figura 14.14 puede utilizarse con muchos microcontroladores PIC de18 pines y el objetivo siempre será el de comunicarnos con una computadora por medio de su puerto serial, que hará las veces de equipo DTE y quien deberá estar ejecutando un programa terminal que permita monitorear todos los datos que transmita o reciba por este puerto.

Si el lector no tiene amplios conocimientos en algún lenguaje de programación, no hay problema: Windows ofrece el programa *hyperterminal* que lo podemos utilizar fácilmente.

```
;-----LIB1.GRP-----------------------------
;ESTE PRIMER MÓDULO RECIBE UN CARÁCTER DE 8
;BITS, 1 DE INICIO, 1 DE
;PARADA Y SIN PARIDAD
;RESULTADO FINAL: CARÁCTER RECIBIDO EN EL
;REGISTRO REGREC
;-----------------------------------

REC_ASINC
      BCF   BANDERA,1  ; LIMPIA BANDERA DE
                       ; ERROR DE INICIO
      BCF   BANDERA,0  ; LIMPIA BANDERA DE
                       ; ERROR DEL PAQUETE
      MOVLW 09         ; 8 BITS DE DATOS + 1
                       ; DE PARADA
      MOVWF CUENTA_BIT

REC_ASINC1
      BTFSC PORTA,RECE ; LLEGO BIT DE
                       ; INICIO?
      GOTO  REC_ASINC1 ; NO -> REGRESO
      CALL  MEDIO_BIT  ; SI -> ESPERO UN
                       ; TIEMPO DE 1/2 BIT
      BTFSS PORTA,RECE ; ¿AÚN SE TIENE EL BIT
                       ; DE INICIO?
      GOTO  REC_ASINC2 ; SI -> SALTO A
                       ; REC_ASINC2
      BSF   BANDERA,1  ; NO -> ACTIVO BIT 1
                       ; DE BANDERA
      GOTO  REC_ASINC_FIN ;Y SALGO

REC_ASINC2
      CALL  BIT        ;ESPERO EL TIEMPO DE 1 BIT
      DECFSZ CUENTA_BIT,F ; ACTUALIZO EL
                       ; CONTADOR E BITS
      GOTO  REC_ASINC3 ; SI NO HA TERMINADO
                       ; EL PAQUETE SALTO.
      BTFSS PORTA,RECE ; ¿BIT DE STOP LISTO?
      BSF   BANDERA,0  ; NO -> ACTIVO BIT 0
                       ; DE BANDERA
      GOTO  REC_ASINC_FIN ; Y SALGO

REC_ASINC3
      BSF   ESTADO,C   ; ACTIVO EL CARRY
      BTFSS PORTA,RECE ; TOMO EL BIT DE
                       ; DATO. ¿ES 1?
      BCF   ESTADO,C   ; NO -> BORRO CARRY
      RRF   REGREC,F   ; ROTO EL CARRY CON EL
                       ; REGISTRO DE RX
      GOTO  REC_ASINC2 ; SALTO PARA ESPERAR
                       ; EL SIGUIENTE BIT

REC_ASINC_FIN
      RETLW 0          ; SALIDA


;-----------------------------------
; ESTE SEGUNDO MÓDULO ENVÍA UN CARÁCTER DE 8
; BITS, 1 DE INICIO, 1 DE PARADA
; SIN PARIDAD
; ENTRA: CARÁCTER A SER ENVIADO EN EL
; REGISTRO REGTX Y PORTA,1 DEBE
; ESTAR EN ALTO
; AL SALIR EL PUERTO DE TX DEBE QUEDAR EN
; ALTO.
;-----------------------------------

TRA_ASINC
      BCF   PORTA,TX ; BIT DE INICIO
      CALL  BIT      ; ESPERO EL TIEMPO DE 1
BIT
      MOVLW 08       ;8 BITS DEL CARÁCTER
      MOVWF CUENTA_BIT ;SE CARGA EN ESTE
                     ; REGISTRO
TRA_ASINC1
      RRF   REGTX,F  ;ROTA BIT A TRAV…S DEL
                     ;CARRY
      BTFSC ESTADO,C ;CARRY = 0 ?
      GOTO  TRA_ASINC2 ;NO -> C  = 1, PA1 =1
      BCF   PORTA,TX ;BIT DE DATOS = 0.
                     ;PA1= 0
      GOTO  TRA_ASINC3 ;Y SALTO
TRA_ASINC2
      BSF   PORTA,TX ;BIT DE DATOS = 1,
                     ;PA1 = 1

TRA_ASINC3
      CALL  BIT      ;ESPERA EL TIEMPO DE 1 BIT
      DECFSZ      CUENTA_BIT,F ;¿YA SE ENVIARON
                     ;LOS 8 BITS?
      GOTO  TRA_ASINC1 ;NO -> IR POR EL
                     ;SIGUIENTE BIT
      BSF   PORTA,TX ;SI -> HACER EL BIT
                     ;DE PARADA
      CALL  BIT      ;ESPERA EL TIEMPO DE 1 BIT
      RETLW 0

;-----------------------------------
; RETARDO UTILIZADO PARA TEMPORIZAR LOS BITS
;-----------------------------------
RET   MOVF        CONTAB,W
      MOVWF       CONTAC
RET1  MOVF        CONTAA,W
      MOVWF       CONTAD
RET2  DECFSZ      CONTAD,F
      GOTO        RET
      DECFSZ      CONTAC,F
      GOTO        RET1
      RETLW       0

MEDIO_BIT
      MOVLW       D'3'
      MOVWF       CONTAB
      MOVLW       D'20'
      MOVWF       CONTAA
      CALL        RET
      RETLW       0

BIT   MOVLW       D'3'
      MOVWF       CONTAB
      MOVLW       D'43'
      MOVWF       CONTAA
      CALL        RET
      RETLW       0
;-----------------------------------
```

Figura 14.15 *Rutinas de recepción y transmisión no sincrónica*

En la figura se puede apreciar que el microcontrolador se conecta a la computadora por medio de un *transceiver*, conformado por el circuito integrado MAX232 y los pines de conexión entre este último y un conector para puerto serial tipo DB-25; como elementos de señalización se han conectado diodos LED en los puertos del microcontrolador.

Comunicación *Half-Duplex*

En muchos casos la aplicación de una comunicación *Half-Duplex* es más que suficiente para satisfacer plenamente las exigencias de un sistema en donde se transmiten datos.

Desde el punto de vista del receptor se pueden dar dos casos: el primero, es cuando el microcontrolador debe obligadamente esperar la llegada de un carácter para poder ejecutar sus tareas, caso en el cual podemos hacer que el programa se encierre en un ciclo hasta que llegue el carácter. El segundo caso se presenta cuando el microcontrolador posee un tiempo limitado para esperar el carácter (*timeout*), ya que debe estar pendiente de otras actividades.

Los módulos de *software* mostrados en la **figura 14.15**, conforman las subrutinas de recepción y transmisión. La velocidad de transmisión es de 2.400 bps y cada carácter posee un formato compuesto por 1 *bit* de inicio, 8 *bits* de datos y 1 *bit* de parada, **figura 14.16**.

Figura 14.16 *Formato del carácter*

Figura 14.17 *Registro BANDERA*

Recepción *Half-Duplex*

Con el objeto de tener el control sobre la transmisión, se ha definido un registro de banderas (BANDERA) cuyos *bits* se activan o desactivan de acuerdo a los errores encontrados. Si el microcontrolador detecta un *bit* de inicio cuya duración es menor al tiempo normal de 1/2 *bit*, lo

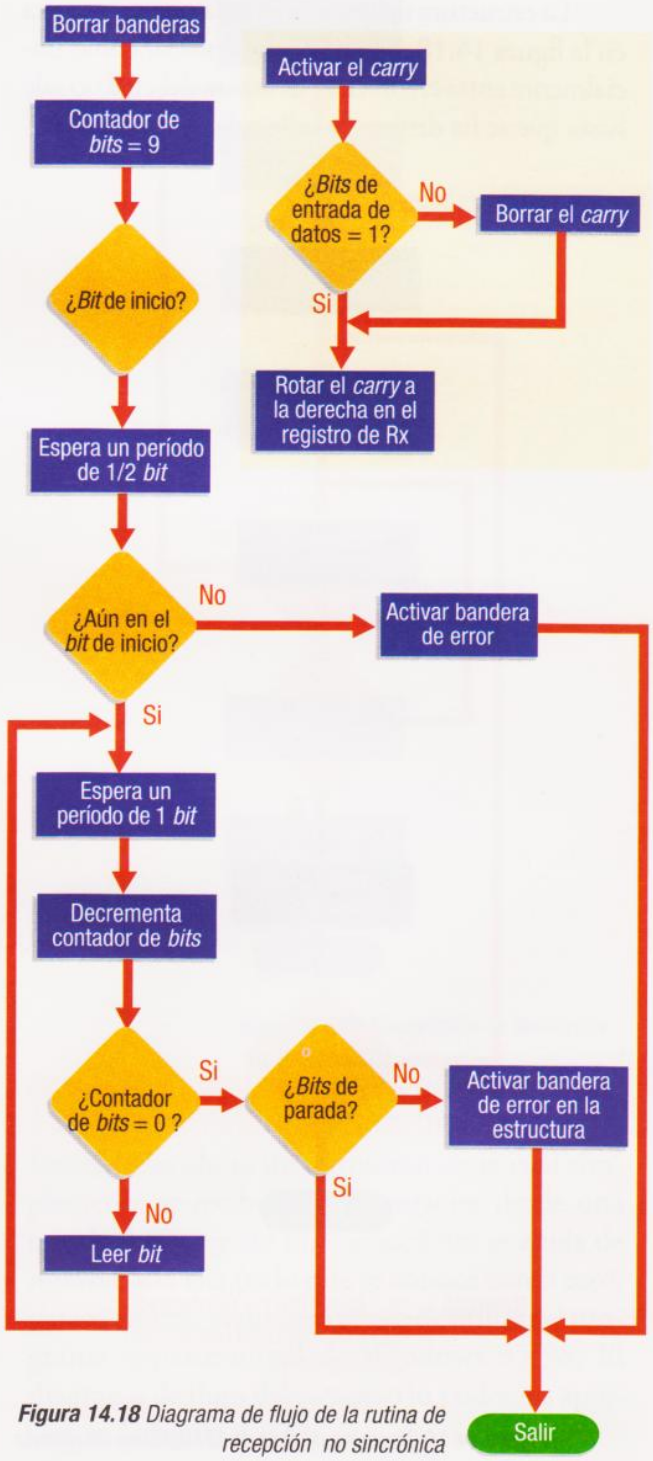


Figura 14.18 *Diagrama de flujo de la rutina de recepción no sincrónica*

más probable es que se trate simplemente de un ruido que apareció en la línea y en ese momento se activa el *bit* 1 de BANDERA. Si por alguna razón el *bit* de parada no llega en el momento exacto que se esperaba, se genera una nueva situación de error y se activa el *bit* 0 de BANDERA, **figura 14.17**.

La estructura del programa es la que se muestra en la **figura 14.18**. Allí podemos apreciar como inicialmente entra en un ciclo de espera del cual no sale hasta que se ha detectado la llegada del primer *bit*.

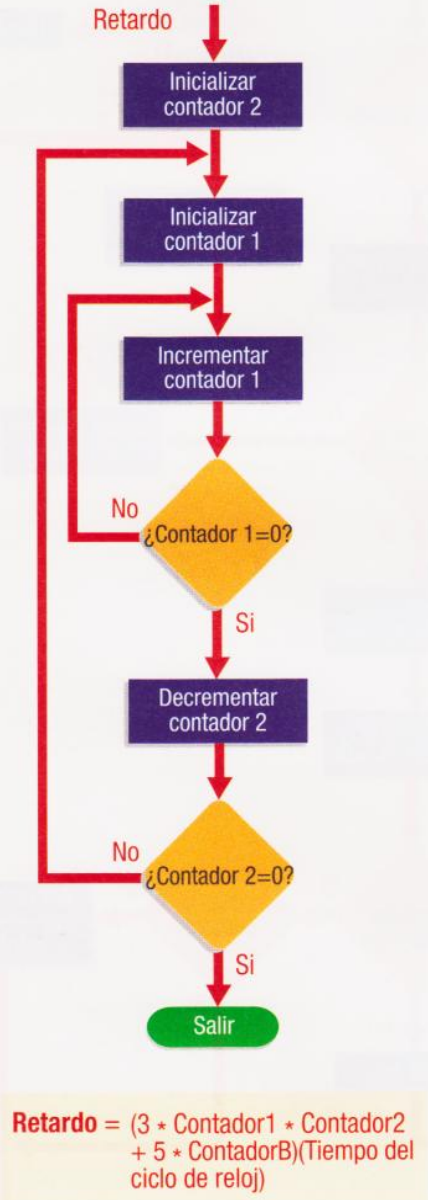


Figura 14.19 *Diagrama de flujo de la subrutina de retardo*

Al detectar la presencia de un cero lógico en la línea, se presume que se trata del *bit* de inicio, se debe esperar un tiempo (1/2 *bit*) y de nuevo comprobar que ese nivel permanece en la línea; si la comprobación no es exitosa se establece un estado de error en la transmisión, activándose el *bit* 1 de BANDERA. Si se ha comprobado la llegada del *bit* de inicio, se espera el tiempo gastado por un *bit* para hacer la primera lectura; en este punto se ha esperado un tiempo de $1^1/_2$ *bits* desde que se detectó el *bit* de inicio hasta que se leyó el primer *bit*. A partir de ese momento se hacen 9 lecturas, cada una separada por el tiempo gastado por un *bit*. Si la novena lectura es un uno lógico se asume una lectura correcta y el valor leído se coloca en el registro REGREC. Si el noveno *bit* no corresponde a un uno, se configura un estado de error denominado error en la estructura y se activa el *bit* 0 del registro BANDERA.

El registro de recepción toma el dato de la transmisión por medio de un procedimiento en el cual el *bit* leído se coloca en el *bit carry* del registro de estado y se hace una rotación del registro REGREC.

La subrutina de retardo la podemos apreciar en la **figura 14.19**. Se compone de un ciclo embebido dentro de otro ciclo que se repi-

Rata Bits/s	Tiempo de 1 *bit* (µ s)	Tiempo 1/2 *bit* (µ s)
110	9090,89	4545,5
300	3333,3	166,7
600	1666,6	833,3
1200	833,3	416,7
2400	416,7	208,3
4800	208,3	104,2
9600	104,2	52,1
19200	52,1	26

Tabla 14.4 *Tiempos empleados para distintas velocidades de transmisión*

te un cierto número de veces decrementado algunos registros. En la **tabla 14.4** se muestran los distintos periodos de tiempos que emplean un *bit*, y 1/2 de *bit* para las distintas velocidades de transmisión.

La instrucción RRF: rote el contenido del registro "f" a la derecha, usando el carry

Sintaxis: RRF f,d
Los *bits* del registro f son desplazados una casilla hacia la derecha, y el *bit* que ocupaba la posición cero es ahora el *bit carry*. Si d=0 el resultado final se guarda en W, si d=1 el resultado se guarda en f.

Transmisión *Half-Duplex*

Para desarrollar esta función se debe tener el cuidado de manejar la misma velocidad y formato de los caracteres que los utilizados en la recepción, pero en este caso no se han establecido banderas de error.

El dato que se va a enviar serialmente debe estar almacenado en un registro denominado registro de transmisión. La transmisión debe empezar colocando un cero lógico por la línea de salida que corresponde al *bit* de inicio, luego espera el tiempo correspondiente a un *bit* para que el tiempo de dicho *bit* sea el apropiado y a partir de ese momento se hacen ocho rotaciones (a través del *carry*) del registro de transmisión, **figura 14.20**.

Cada vez que se hace una rotación a la derecha del registro de transmisión, cada uno de los *bits* de datos empieza a circular secuencialmente por el *carry*. De esta manera solo necesitamos chequear el estado de dicho *bit* para saber el estado que debe tomar la línea de transmisión en cada instante de tiempo, **figura 14.21**.

Figura 14.20 *Transmisión no sincrónica*

Aplicación de las rutinas de recepción y de transmisión no sincrónicas

Estudiemos ahora una situación en la cual simplemente se recibe un *byte* carácter desde una terminal (PC) y ese mismo carácter se envía de nuevo hacia ella (es lo que se conoce como eco); para visualizar el proceso se puede utilizar el programa hyperterminal de Windows 95/98. El diagrama de flujo del ejercicio lo podemos apreciar en la **figura 14.22**.

Figura 14.21 *Estableciendo el dato de la línea de transmisión*

Figura 14.22 *Ejercicio de comunicación*

Cuando el sistema está listo para recibir un carácter, activa la línea de DTR (*Data Terminal Ready*); al recibir el dato, el microcontrolador deshabilita la línea DTR (PA2) y envía el mismo carácter hacia el terminal. Deshabilitar DTR equivale a decirle al terminal que no envíe más caracteres porque el microcontrolador está ocupado y no va a recibir más datos porque ahora él es quien va a transmitir.

Este es un ejemplo muy sencillo de control del flujo de la comunicación en una transmisión *Half-Duplex*; si el microcontrolador recibe un *bit* de error en el registro bandera, retorna un carác-

```
;------SERIAL1.ASM-----------------
;EJERCICIO DE COMUNICACIÓN SERIAL
; 2400 BPS, 1 BIT DE INICIO, 1 DE PARADA
; ENTRADA DE DATOS: PA3
; SALIDA DE DATOS : PA1,   DTR: PA2
      RADIX HEX
      LIST  P=16F84
C           EQU  0
Z           EQU  2
W           EQU  0
F           EQU  1
RECE        EQU  3
TX          EQU  1
ESTADO      EQU  03
PORTA       EQU  05
TRISA       EQU  85
PORTB       EQU  06
TRISB       EQU  86
BANDERA     EQU  0C
CUENTA_CHAR      EQU  1C
CUENTA_BIT EQU  1D
CONTAA      EQU  1E
CONTAB      EQU  1F
CONTAC      EQU  0D
CONTAD      EQU  0F
REGREC      EQU  10
REGTX       EQU  11
      ORG   0
INICIO
      CLRF  PORTA
      BSF   ESTADO,5   ;SALTA AL BANCO 1
      MOVLW B'00001111'
      MOVWF TRISA
      MOVLW 0X00
      MOVWF TRISB
      BCF   ESTADO,5   ;SALTA AL BANCO 0
      BSF   PORTA,2    ;DESACTIVA DTR
      BSF   PORTA,1    ;SALE ESTADO DE MARCA
LAZO  CLRF  BANDERA
      BCF   PORTA,2    ;ACTIVA DTR
      CALL  REC_ASINC
      BSF   PORTA,2    ;DESACTIVA DTR
      CALL  BIT        ;ESPERA UN PERIODO DE 1 BIT
      BTFSS BANDERA,0  ;¿ERROR EN LA ESTRUCTURA?
      GOTO  LAZO1
      MOVLW 10         ;NAK
      MOVWF REGREC     ;Y LO DEVUELVE
LAZO1 MOVF  REGREC,W
      MOVWF REGTX
      CALL  TRA_ASINC
      GOTO  LAZO
      INCLUDE    LIB1.GRP
      END
```

Figura 14.23 *Código fuente del ejercicio de comunicación serial*

ter ASCII indicador de que reconoció el error. En las redes de comunicación, cuando el emisor se ha enterado de este carácter, puede tomar acciones tales como enviar nuevamente el dato.

El código fuente del ejercicio lo podemos apreciar en la **figura 14.23** El microcontrolador habilita la línea DTR para informar que está listo para recibir datos y llama la rutina REC_ASINC para tomar el carácter. Al hacer la recepción y si no se ha activado ninguna bandera de error, el dato pasa del registro de recepción REGREC al registro de transmisión REC-TX y se transmite.

Recordemos que la rutina de recepción en realidad lee el *bit* de parada cuando éste aún está en la mitad de su periodo. Para evitar conflictos es conveniente esperar un periodo de un *bit* después de que se ha deshabilitado la línea DTR; de esta manera evitaremos algunos riesgos como por ejemplo, que el microcontrolador envíe datos cuando el terminal aún esta emitiendo el *bit* de parada.

Comunicación serial *Full-Duplex*

Este tipo de comunicación se caracteriza por desarrollar simultáneamente los procesos de recepción y transmisión. Debido a que los microcontroladores son dispositivos que ejecutan una serie de instrucciones de manera secuencial a través del tiempo, este tipo de comunicación debe implantarse dividiendo el proceso de recepción y transmisión en distintas tareas. Como hemos visto anteriormente, la unidad básica de transmisión dentro de las comunicaciones no sincrónicas es el *bit*; si el tiempo empleado por cada una de estas unidades se divide en pequeños intervalos de tiempo, podemos hacer una buena simulación de operaciones simultáneas de transmisión y recepción. Para ello, simplemente se ejecuta la tarea de recepción en un instante de tiempo y la tarea de transmisión en el siguiente instante; este modo de trabajo es viable siempre y cuando se tenga un estricto cuidado en el control de los tiempos, el cual se hace mediante subrutinas de retardo basadas en ciclos anidados de programa. En la **figura 14.24** podemos apreciar el manejo que se debe hacer del tiempo de recepción y transmisión; allí podemos observar como el tiempo de cada *bit* se ha dividido en pequeños intervalos que sirven para definir el instante exacto en que deben operar cada una de las tareas. Observemos como la transmisión se lleva a cabo sólo a partir de que la recepción ha reconocido satisfactoriamente cada uno de los *bits* de datos, mostrando una especie de intercalado en la ejecución de ambos procesos.

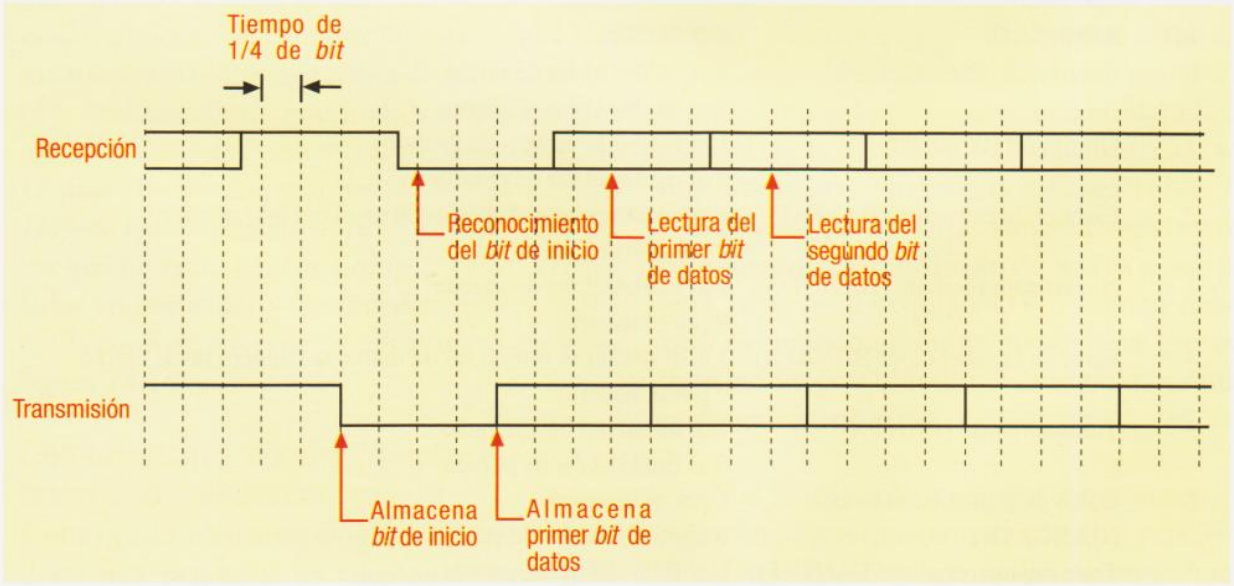


Figura 14.24 *Temporización de las tareas*

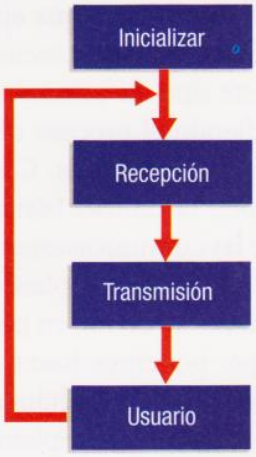


Figura 14.25 *Lazo de tareas*

Una segunda posibilidad para hacer la distribución de tareas en instantes de tiempo es configurando una interrupción del temporizador, bajo la cual se ejecutan secuencialmente la recepción y la transmisión.

Estudiemos un programa de ejemplo donde podamos visualizar mejor estos esquemas. Se compone de tres partes o tareas principales: la tarea de recepción, la tarea de transmisión y la tarea del usuario. Todas ellas se ejecutan secuencialmente por estar dentro de un ciclo que se cumple indefinidamente, **figura 14.25**. Para facilitar el desarrollo de este programa se creó primero un archivo (full.grp) con las distintas subrutinas que se requieren para transmitir y recibir estilo *Full-Duplex*.

Para controlar los distintos estados de la comunicación se ha establecido un registro banderas denominado BANCOM, cuya distribución se muestra en la parte de abajo.

En la **figura 14.26** podemos apreciar el código del programa que desarrolla el método de las tareas para distintos instantes de tiempo, el cual podemos incluir dentro de nuestro proyecto particular. En este programa podemos distinguir claramente la manera como se construyeron las dos tareas. Existe una tarea adicional denominada la tarea del usuario. Ella contiene, como su nombre lo sugiere, los procedimientos específicos que él requiere para su proyecto particular.

En cualquier programa que utilice estas subrutinas, la tarea del usuario debe primero chequear la existencia de nuevos datos recibidos (bandera BANUDA) y si se ha recibido un nuevo carácter debe moverlo hacia el registro de datos recibidos; de esta manera, el registro de recepción del usuario queda disponible para nuevas lecturas.

BIT	BANDERA DE	INDICACIÓN
0	Recibiendo (BANRECE)	1 al recibir el *bit* de inicio 0 al recibir el *bit* de parada
1	Último *bit* (BANULB)	1 al recibir el último *bit* de datos 0 al recibir el *bit* de parada
2	Error en la estructura (BANEST)	1 si no se recibió *bit* de parada en el décimo periodo 0 por el usuario
3	Nuevo dato recibido (BANUDA)	1 al recibir el *bit* de parada 0 por el usuario
4	Error en la recepción (BANRES)	1 si al recibir el último *bit* de datos, la bandera no se activó 0 por el usuario
5	Transmitiendo (BANTX)	1 al inicio del *bit* de inicio 0 al final del *bit* de parada
6	Datos de transmisión listos (BANDATR)	1 por el usuario 0 al inicio del *bit* de inicio
7	Transmisión completa (BANTXF)	1 al final del *bit* de parada 0 por el usuario

```
;=====FULL.GRP=======
;-----------------
;RECEPCION
;-----------------
RETA    SET     RETA1
FREC    BTFSS   BANCOM,BANRECE ;
        GOTO    FREC2
        DECFSZ  LAZORECE

        GOTO    ET2
        BTFSC   BANCOM,BANULB ; BANDERA DE ULTIMO BIT?
        GOTO    FREC3         ; SI -> IR A FREC3
        BSF     ESTADO,C      ; NO -> LEER BIT
        RRF     REGREC,F      ; ROTAR CARRY EN EL
                              ; BYTE DE RX
        MOVLW   04            ; CARGO CONTADOR
        MOVWF   LAZORECE
        DECFSZ  ROTABIT       ; ACTUALIZA CONTADOR.
                              ; SALTA SI SON OCHO.
        GOTO    RETA1         ; SI NO-> IR A RETARDO
        BTFSS   BANCOM,BANUDA ; BANDERA DE NUEVO DATO
                              ; RECIBIDO?
        BSF     BANCOM,BANRES ;
        BSF     BANCOM,BANULB ; ACITVAR BANDERA DE
                              ; ULTIMO BIT
        MOVF    REGREC,W      ; MOVER BYTE RECIBIDO AL
                              ; REGISTRO DEL
                              ; RECEPCION DEL USUARIO.
        MOVWF   RECE          ;
        GOTO    RETA2         ; IR A RETARDO

FREC2
        BTFSC   PTOENT,PINENT ; BIT DE INICIO?
        GOTO    RETA3         ; NO-> RETARDO
        BSF     BANCOM,BANRECE ; SI ->ACTIVAR BANDERA
                              ; DE RECEPCION
        MOVLW   08            ; INICILIZAR CONTADOR DE
                              ; BITS DE RECEPCION
        MOVWF   ROTABIT
        MOVLW   05
        MOVWF   LAZORECE      ; INICIALIZR CONTADOR
                              ; DE LAZOS DE RX
        GOTO    RETA4         ; IR A RETARDO

FREC3
        BTFSC   PTOENT,PINENT ; BIT DE PARADA?
        GOTO    FREC4
        BSF     BANCOM,BANEST ; NO->ACTIVAR BANDERA
                              ; DE ERROR EN ESTRUCTURA
FREC4
        BCF     BANCOM,BANRECE; LIMPIAR BANDERA DE
                              ; RECEPCION
        BCF     BANCOM,BANULB ; BORRAR BANDERA DE
                              ; ULTIMO BIT RECIBIDO
        BSF     BANCOM,BANUDA ; ACTIVAR BANDERA DE
                              ; NUEVO DATO RECIBIDO
        GOTO    RETA          ; E IR A RETARDO
ET2     NOP
RETA3   GOTO    R1
R1      GOTO    R2
R2      GOTO    RETA4
```

```
RETA4   NOP
        GOTO    RETA1
RETA1   GOTO    R3
R3      GOTO    R4
R4      GOTO    RETA2
RETA2   NOP
;-----------------
; TRANSMISION
;-----------------
TXON    BTFSS   BANCOM,BANTX ; TRANSMITIENDO?
        GOTO    TXON2        ; NO-> IR A TXON2
        DECFSZ  LAZOTX,F     ; SI-> ACTUALIZAR
                             ; CONTADOR DE LAZO
SALTO   GOTO    RETA5        ; BIT AUN NO ESTA
                             ; COMPLETO. IR A RETARDO
        MOVLW   04
        MOVWF   LAZOTX       ; TIME OUT. INICIALIZAR
                             ; CONTADOR DE LAZO
        DECFSZ  CNTBITX,F    ; ACTUALIZAR CONTADOR DE BIT
        GOTO    TXON4        ; NO SE HAN ENVIADO TODOS
                             ; LOS BITS-->IR A
                             ; SIGUIENTE BIT
        BCF     BANCOM,BANTX ; SE HAN ENVIADO TODOS LOS BITS.
        BSF     BANCOM,BANTXF; ACTIVAR BANDERA DE TX COMPLETA
        GOTO    RETA6

TXON2   BTFSS   BANCOM,BANDATR; DATOS DE LA TX LISTOS?
        GOTO    RETA7        ; NO -> RETARDO
        BSF     BANCOM,BANTX ; DATOS LISTOS. ACTIVAR BANDERA
        MOVLW   0A
        MOVWF   CNTBITX      ; INICIALIZAR CONTADOR DE BITS
        MOVLW   04
        MOVWF   LAZOTX       ; INICIALIZAR CONTADOR DE LAZOS
        MOVF    REGDAT,W     ; MOVER DATO DEL USUARIO
                             ; AL REGISTRO DE TX
        BCF     BANCOM,BANDATR; LIMPIAR BANDERA DE
                             ; DATOS DE TX LISTOS

ET3     BCF     PTOSAL,PINSAL; BIT DE INICIO
        GOTO    TXON8
TXON4   BSF     ESTADO,C
        RRF     REGTX,F      ; MOVER BIT DE DATOS AL CARRY
TXON5   BTFSC   ESTADO,C     ; CARRY = 0 ?
        GOTO    TXON6
        BCF     PTOSAL,PINSAL ; SI -> BIT DE DATO = 0 ,
                             ; LIMPIAR PIN PA1
        GOTO    TXON7
TXON6   BSF     PTOSAL,PINSAL ; PONER EN UNO PIN PA1
        NOP
TXON7   GOTO    TXON8        ; FINALIZAR LA TX
RETA5   NOP
RETA7   GOTO    ET4
ET4     GOTO    ET5
ET5     GOTO    RETA6
RETA6   GOTO    ET6
ET6     GOTO    ET7
ET7     GOTO    TXON8
TXON8   NOP                  ; SALIR DE LA TRANSMISION
LASO    DECFSZ  CONRET,F
        GOTO    LASO
        RETLW   0
```

Figura 14.26 *Subrutinas de recepción y transmisión*

```
;=========================================
;==== COMUNICACION SERIAL NO SINCRONICA=====
;=============== FULL DUPLEX================

        RADIX  HEX
        LIST P = 16F84
C       EQU    0
Z       EQU    2
DC      EQU    1
PD      EQU    3
TO      EQU    4
RP0     EQU    5
RP1     EQU    6
W       EQU    0
F       EQU    1
; REGISTROS DE LA CPU
;-----------------------
INDF    EQU    00
TMR0    EQU    01
PCL     EQU    02
ESTADO EQU     03
FSR     EQU    04
PORTA   EQU    05
PORTB   EQU    06
EEDATA EQU     08
EEADR   EQU    09
PCLATH EQU     0A
INTCON EQU     0B
OPTREG EQU     01
TRISA   EQU    85
TRISB   EQU    86
EECON1 EQU     08

; MEMORIA DE TRABAJO
;-----------------------
BANCOM EQU     0C
CONRET EQU     0D
REGREC EQU     0E
RECE    EQU    0F
LAZORECE EQU   10
ROTABIT EQU    11
REGTX   EQU    12
REGDAT EQU     13
LAZOTX EQU     14
CNTBITX EQU    15
BANUSU EQU     16
;BITS DEL REGISRO BANCOM
BANRECE        EQU    00;BANDERA DE RECEPCION
BANULB EQU     01;BANDERA DE ULTIMO BIT
BANEST EQU     02;BANDERA DE ERROR EN LA ESTRUCTURA
BANUDA EQU     03;BANDERA DE NUEVO DATO RECIBIDO
BANRES EQU     04;SOBEFLUJO EN LA RX
BANTX  EQU     05;BANDERA DE TRANSMISION
BANDATR EQU    06;BANDERA DE DATOS LISTOS PARA TX
BANTXF EQU     07;BANDERA DE TRANSMISION COMPLETA
PTOSAL EQU     PORTA
PINSAL EQU     01
PTOENT EQU     PORTA
PINENT EQU     03

        ORG    0
INICIO CLRF    PORTA
        BSF    PTOSAL,PINSAL; ENVIAR MARCA
        CLRF   PORTB
        BSF    ESTADO,RP0
        MOVLW  00E9
        MOVWF  TRISA; CONFIGURANDO PINES PTO A
        BCF    ESTADO,RP0
        CLRF   BANCOM
        BSF    BANCOM,BANTXF; ACTIVAR BANDERA DE TX
                              ; COMPLETA
        GOTO   FREC
        include        FULL.GRP
;-----------------------
; TAREA DEL USUARIO
;-----------------------

ECO     BTFSS  BANCOM,BANUDA ; NUEVO DATO RECIBIDO?
        GOTO   RET7
        MOVF   RECE,W ; SI-> TOMAR LOS DATOS
                      ; RECIBIDOS
        MOVWF  REGDAT ; Y LLEVARLOS AL REGISTRO DE TX
                      ; DEL USUARIO
        BCF    BANCOM,BANUDA; BORRAR BANDERA DE NUEVO
                      ; DATO RECIBIDO
        BTFSS  BANCOM,BANTXF; TX COMPLETA?
        GOTO   RETA8
        BCF    BANCOM,BANTXF; BORRAR BANDERA DE FIN
                      ; DE TX
        BSF    BANCOM,BANDATR; ACTIVAR BANDERA DE
                      ; DATOS DE TX LISTOS
        GOTO   RETA9
RET7    GOTO   RETA8
        GOTO   Q1
Q1      NOP
RETA8   NOP
        GOTO   RETA9
RETA9   MOVLW  D'14'
        MOVWF  CONRET
        CALL   LASO
        GOTO   Q3
Q3      GOTO   FREC
        END
```

Figura 14.27 Código fuente del ejercicio

De la misma manera, el usuario no debe mover ningún dato al registro de transmisión hasta que esta tarea haya activado la bandera de transmisión completa (BANTXF). Todas las banderas se encuentran disponibles para el usuario; algunas de ellas son activadas o desactivadas por las distintas tareas, pero otras requieren de la intervención del programador para modificarlas.

Ejemplo de aplicación de las subrutinas

El programa que podemos apreciar en la **figura 14.27** aplica las subrutinas del programa anterior y

con ellas se encarga de recibir un carácter, proveniente de un terminal y lo devuelve (envía un eco) hacia este último. Si el terminal envía continuamente caracteres, los cuales se devuelven de la misma manera, se configura un sistema funcionando como *Full-Duplex*. El formato del carácter para este ejemplo es de ocho *bits* de datos, sin *bit* de paridad y un *bit* de parada; los datos viajan a una velocidad de 2.400 bps, si el cristal es de 4MHz.

Como se anotó anteriormente, el programa hace un lazo cuyo tiempo de ejecución equivale al tiempo de duración de un *bit* dividido entre cuatro. Si estamos trabajando a una velocidad de 2.400 bps, cada *bit es de* 4126,66 microsegundos y por tanto el lazo se debe demorar 104,17 microsegundos, es decir 104 ciclos de instrucción (si se trabaja a 4MHz). La tarea del usuario tiene un número limitado de ciclos, de acuerdo a los empleados por las otras dos tareas; en la **Tabla 14.2** se muestran algunos resultados que nos dicen cuantos ciclos se emplean para la tarea del usuario.

Como se puede ver en la **figura 14.28**, la tarea del usuario indaga por la existencia de nuevos datos recibidos; si no los hay, espera un tiempo hasta que se completa el número de ciclos requerido y luego entrega el control a la tarea de recepción mediante la instrucción **goto frec**.

Cristal MHz	Rata de *bits*	Ciclos por lazo de tareas	Ciclos de las tareas del usuario
4	2400	104	62
4	4800	52	10
4	9600	26	4
8	2400	208	164
8	4800	104	62
8	9600	52	10
12	9600	78	36
16	9600	104	62
20	9600	130	88

Tabla 14.2 *Ciclos para la tarea del usuario*

Si se han recibido nuevos datos, el carácter recibido se mueve desde el registro de datos recibidos del usuario hacia un registro denominado registro de datos de transmisión y se desactiva la bandera **BANUDA**(BAndera de NUevo DAto recibido) indicando al usuario que se ha tomado el nuevo carácter. Si el transmisor no se encuentra ocupado, se activa la bandera de datos de transmisión listos (**BANDATR**) lo que le indica a la subrutina de transmisión que puede empezar su trabajo. Después de esto el control del programa pasa a la tarea de recepción.

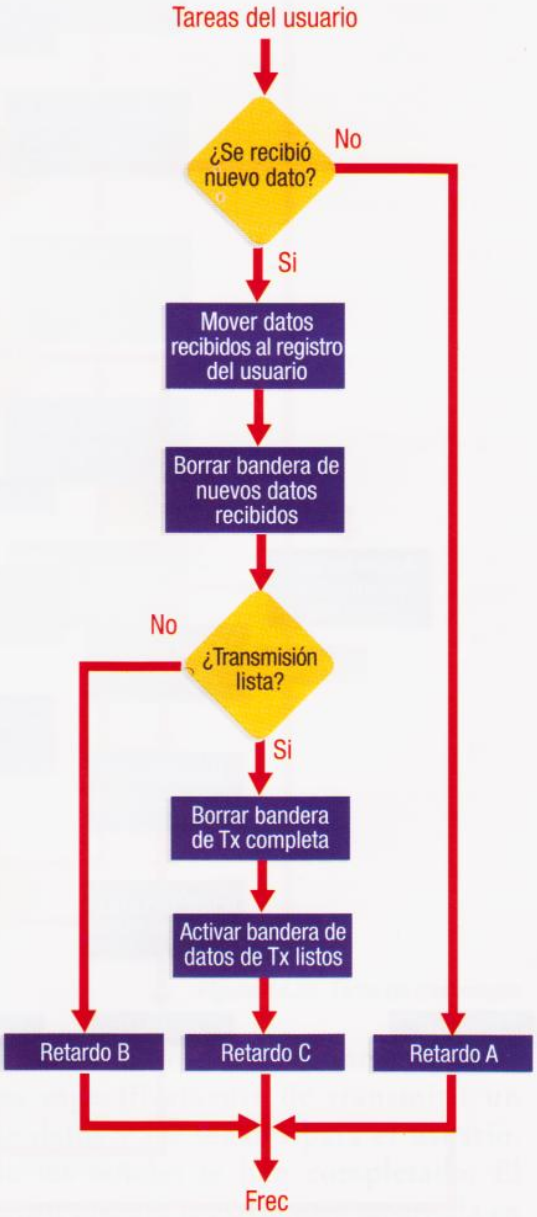


Figura 14.28 *Subrutina o tarea del usuario*

La tarea de recepción, **figura 14.29**, recibe el dato que llega y las señales que le indican al usuario que ha llegado un carácter y que se encuentra disponible para su uso. Después de detectar una condición de *bit* de inicio, la tarea espera cinco miniperiodos de *bit* antes de leer el primer *bit* de datos. Los demás *bits* de datos son leídos cada cuarto miniperiodo de *bit*.

Esta subrutina reconoce y señaliza por medio de banderas dos tipos de error: si el décimo *bit* (*bit* de parada) no se encuentra en estado de marca (1 lógico), entonces se activa el

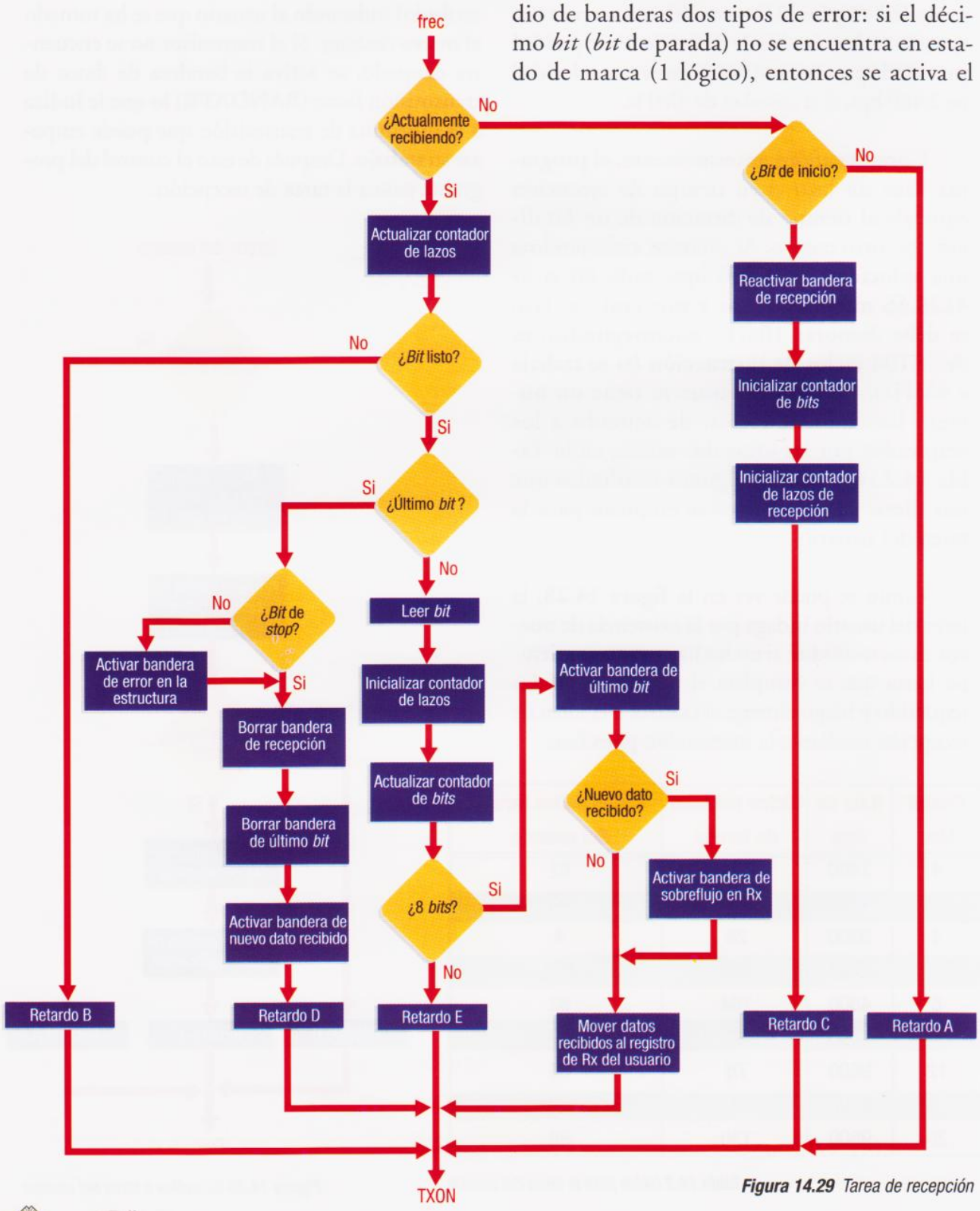


Figura 14.29 *Tarea de recepción*

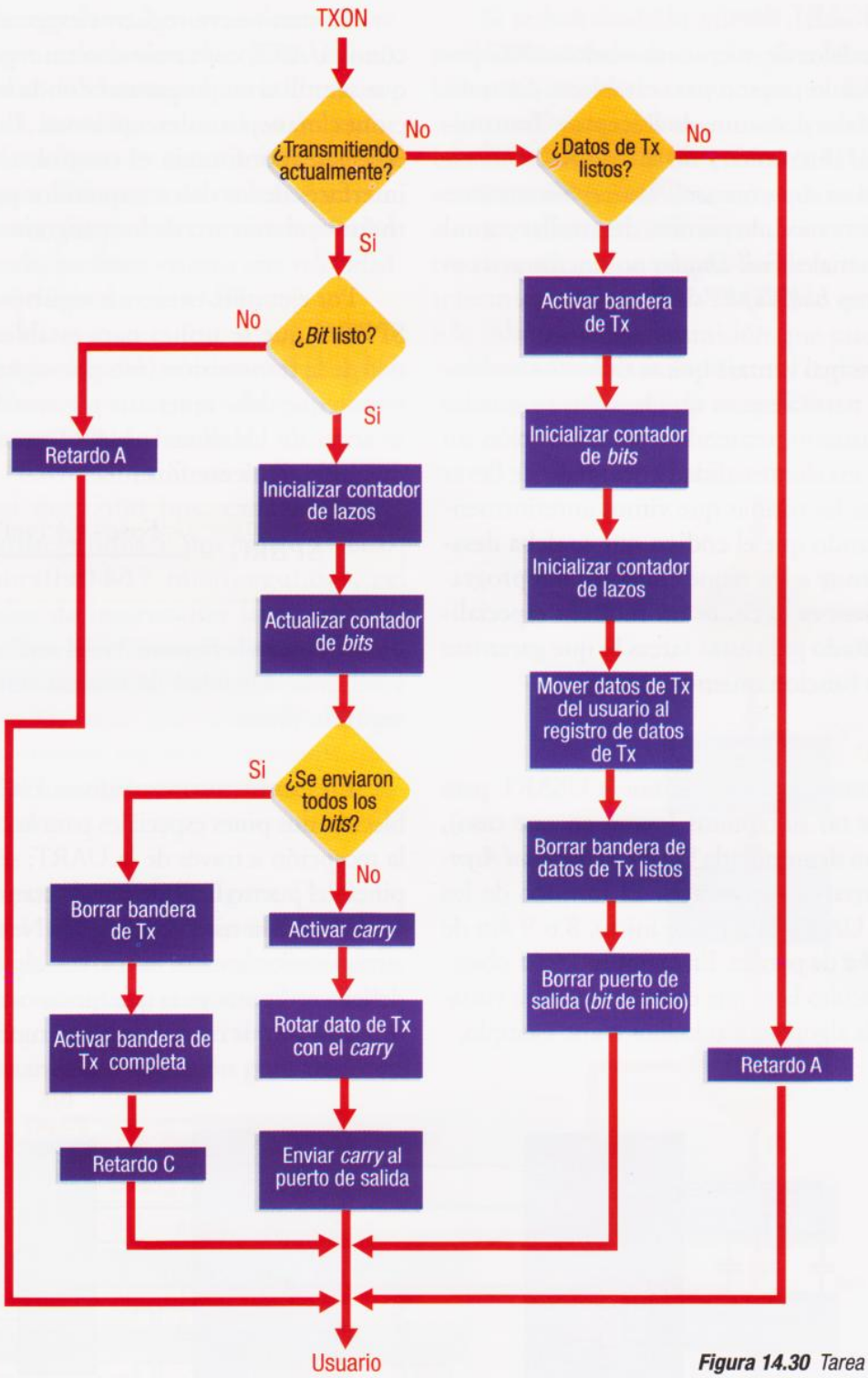


Figura 14.30 *Tarea de transmisión*

bit de error en la estructura (**BANEST**); y si la bandera de nuevos datos recibidos no está borrada cuando el noveno *bit* (último *bit* de datos) se ha leído, se activa la bandera de sobreflujo en la recepción. Ambas banderas deben ser limpiadas por el programador.

La subrutina o tarea de transmisión se encarga específicamente de transmitir un *byte* de datos y las señales para el usuario, cuando las señales se han completado. El transmisor ejecuta la estructura mostrada en la **figura 14.30**.

Módulo USART dentro del *chip*

Muchos modelos de microcontroladores PIC poseen un módulo interno para establecer comunicaciones seriales denominado **Receptor/Transmisor universal sincrónico y no sincrónico** (*Universal Synchronous Asynchronous Receiver Transmitter*. USART). Este módulo permite desarrollar comunicaciones seriales *Full-Duplex* no sincrónas o comunicaciones *half duplex* síncronas.

La principal ventaja que se tiene con módulos de esta naturaleza es que le ahorran mucho trabajo al usuario, teniendo a su disposición un módulo de excelente calidad encargado de llevar a cabo todas las rutinas que vimos anteriormente y asegurando que el código que se deba desarrollar se limite a los requerimientos del programa específico; es decir, es un módulo especializado y diseñado para estas tareas lo que garantiza su correcto funcionamiento.

Utilizando la USART

Muchos autores, cuando utilizan la USART para aplicaciones no sincrónicas (como en este caso), acostumbran denominarla UART (*Universal Asynchronous Receiver/Transmitter*). El formato de los datos de la UART es 1 *bit* de inicio, 8 o 9 *bits* de datos y un *bit* de parada. En la **figura 14.31** observamos el circuito base que utilizaremos para visualizar algunos algoritmos tomados como ejemplo.

Existen nueve registros especiales asociados con la UART, contando con un registro especial que se utiliza en programas donde las comunicaciones manejan interrupciones. Estos registros especiales conforman el control, el estado y las interfaces de los datos requeridos para el correcto funcionamiento de los programas.

Por ejemplo, existe un registro denominado **SPBRG** que se utiliza para establecer la velocidad de la transmisión (*bits* por segundo). El contenido que debe tener este registro debe colocarse antes de habilitar la UART y se calcula mediante la siguiente fórmula:

$$\text{SPBRG} = \frac{\text{Fosc} - 64\ (\text{vel})}{64\ (\text{vel})}$$

donde **Fosc** es la frecuencia del oscilador en MHz y vel es la velocidad de trasnmisión en *bits* por segundo (bps).

En los microcontroladores PIC se han establecido dos pines especiales para la transmisión y la recepción a través de la UART; se trata de dos pines del puerto C: RC6 para la transmisión (TX) y RC7 para la recepción (RX). Necesariamente, aunque inicialmente no parece lógico, ambos se deben configurar para que funcionen como pines de entrada (por medio de la instrucción TRISC).

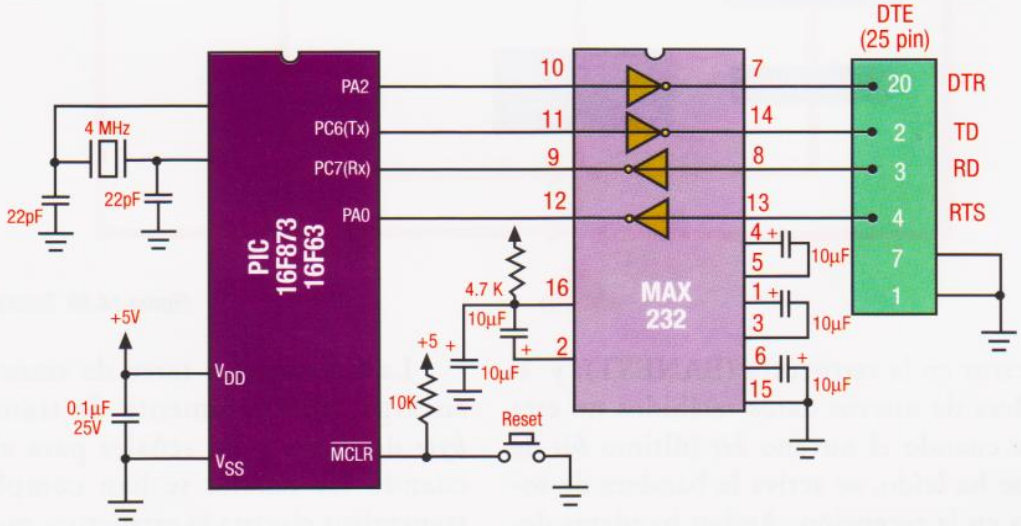


Figura 14.31 Diagrama esquemático del circuito para los ejemplos

Los registros **RCSTA** y **TXSTA** respectivamente, son los que finalmente conforman el estado y control de la recepción y el estado y control de la transmisión, respectivamente.

Los registros **RCSTA, RXSTA** y **PIR1** informan el estado de la comunicación y los registros **RCREC** y **TXREC** son los registros de datos de la recepción y la transmisión, **figura 14.32.**

Los registros **PIE1** e **INTCON** establecen el control de interrupciones en los casos donde se utilicen estas técnicas.

Figura 14.32 *Registros de datos de la UART*

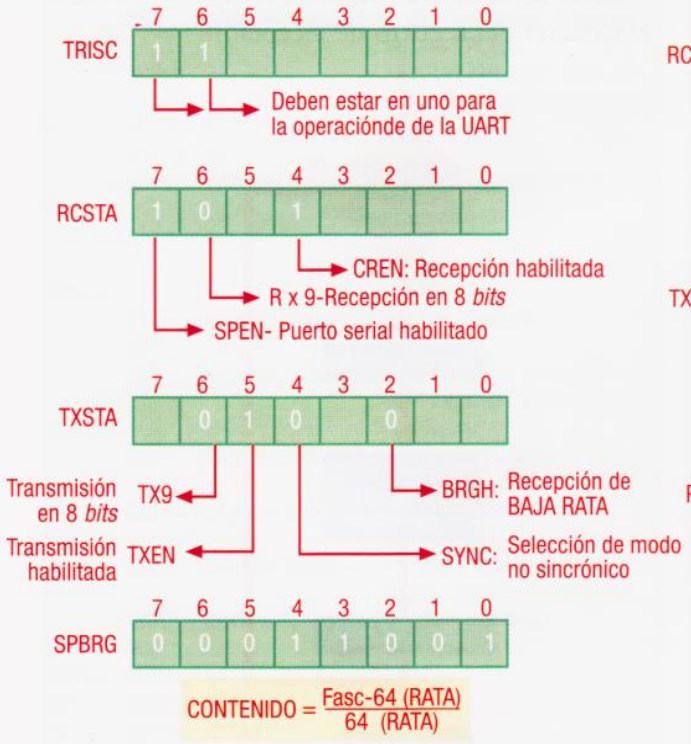


Figura 14.33 *Configuración de los registros de control de la UART*

En la **figura 14.33** se muestran los registros de control configurados para establecer una comunicación con 8 *bits* de datos a 2.400 bps (y un reloj de 4MHz) y en la **figura 14.34** se muestran los registros de control de interrupciones requeridos para configurar una interrupción en la recepción. Finalmente, en la **figura 14.35**, se muestran los *bits* de estado (banderas) de la UART.

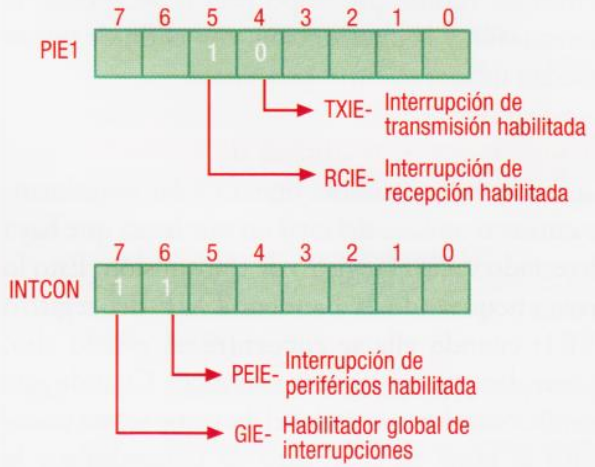


Figura 14.34 *Registros de control para activar una interrupción en la recepción*

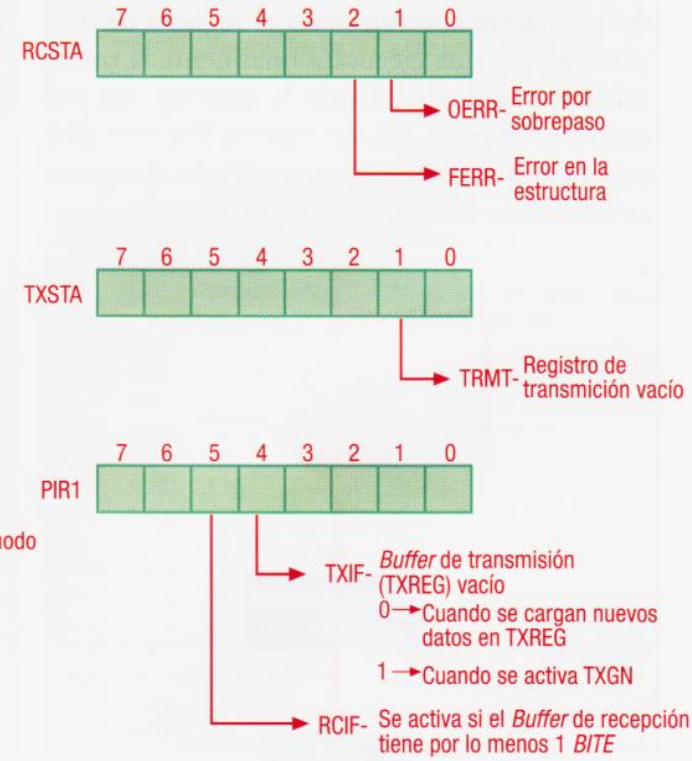


Figura 14.35 *Registros de estado de la UART*

Una característica muy interesante de la UART es que tiene módulos internos en *hardware* completamente independientes para la recepción y para la recepción, lo que permite establecer fácilmente comunicaciones *Full-Duplex*. El proceso finalmente es tan simple como colocar el dato que se va enviar en el registro de transmisión y leer el dato recibido en el registro de recepción; todo lo demás lo hacen los circuitos de la UART. En esta lección daremos las rutinas prototipo para la recepción, la transmisión y la interrupción, con el fin de que se puedan utilizar en otros programas.

Transmisión por medio de la UART

La rutina de transmisión, **figura 14.36**, inicialmente entra en un lazo del cual no sale hasta que haya detectado vacío el registro de transmisión. Esto lo hace chequeando la bandera TXIF del registro PIR1; cuando ella se encuentra en estado alto, quiere decir que el registro está vacío. Cuando esta condición se ha cumplido, el dato que se va a transmitir se pone en el registro de transmisión y la rutina termina.

Un registro vacío indica que ya se ha enviado el carácter anterior hacia el registro de desplazamiento que ejecutará finalmente la transmisión. Si se ha habilitado la interrupción por transmisión (*bit* TXIE del registro **PIE1** en alto) se generará una interrupción cuando el registro se haya desocupado. Si el registro de desplazamiento se encuentra vacío, a él llegan los contenidos del registro de transmisión y se inicia la transmisión del dato. Si el registro de desplazamiento no se encuentra vacío, la transmisión de los contenidos del nuevo registro de transmisión no empieza hasta que el *byte* anterior se haya transmitido completamente.

Figura 14.36 *Transmisión con la UART*

Recepción por medio de la UART

Cuando trabajamos en comunicaciones no sincrónicas, no se puede predecir el momento de llegada de un dato. Al recibir un carácter se puede generar una interrupción notificando este evento, **figura 14.37**. Cuando configuramos la interrupción por recepción en la UART, ella se genera en el momento en que se ha recibido la totalidad del carácter. En ese momento la subrutina de interrupción debe tomar el carácter del registro de recepción y trabajar sobre él dentro de una nueva subrutina, **figura 14.38**.

Lo primero que se debe hacer dentro de esta rutina es comprobar que la interrupción se debió a una recepción, de lo contrario retorna. Dentro de la nueva rutina de recepción se toma el carácter del registro de recepción.

Figura 14.37 *Servicio de interrupción*

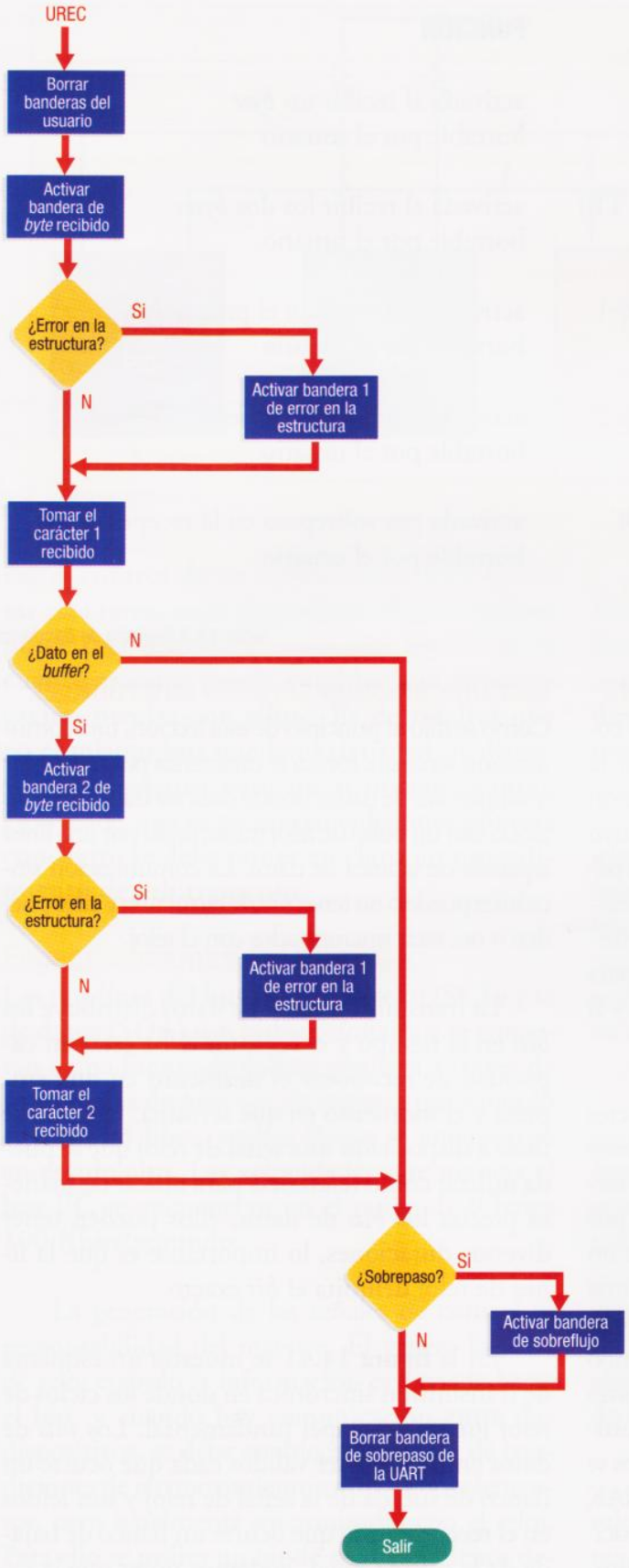


Figura 14.38 *Diagrama de flujo de la recepción por medio de la UART*

El diagrama de flujo de la **figura 14.38** corresponde a una subrutina perteneciente al archivo COM.GRP, el cual podemos incluir dentro de nuestro proyecto particular en cualquier momento, como se mostrará a través de un ejemplo. Esta subrutina trabaja con base en el carácter recibido en el registro de recepción, para asegurarnos que el dato se lee correctamente, como se explica a continuación. Adicionalmente, la misma subrutina se encarga de generar las banderas, **tabla 14.3**, las cuales señalizan el funcionamiento del programa principal. Todas las banderas de esta tabla se encuentran ubicadas dentro del registro **BANREC**.

El registro de recepción que maneja la UART es de dos *bytes* y es del tipo FIFO (***First In First Out***: el primero que entra es el primero que sale). Esto quiere decir que antes de poder leer este registro de recepción se deben haber recibido con anterioridad por lo menos dos caracteres. Si se lee el registro cuando un tercer carácter está llegando, no se perderá la información almacenada. Sin embargo, pero si el registro se lee antes de que hayan llegado los dos caracteres se presentará un error denominado de sobrepaso, en el cual los datos se perderán. Si se presenta esta situación se activará una bandera (OERR) del registro RCSTA.

Los microcontroladores PIC ofrecen una ventaja para no correr riesgos en la lectura del registro de la UART. Cuando el dato del registro está listo y disponible, se activa una bandera (RCIF) del registro PIR1. Al momento de esta activación se debe leer inmediatamente el registro antes de que llegue el tercer carácter y se pueda perder información. En la **figura 14.39** se observa el código fuente de este archivo, el cual usted puede descargar de nuestra página en la internet.

BIT	BANDERA	FUNCIÓN
0	*byte* recibido (NUBYTE)	activada al recibir un *byte* borrable por el usuario
1	dos *bytes* recibidos (DOSBYTE)	activada al recibir los dos *bytes* borrable por el usuario
3	error en la estructura del *byte* 1	activada por error en el primer *byte* borrable por el usuario
4	error en la estructura del *byte* 2	activada por error en el segundo *byte* borrable por el usuario
6	error de sobreflujo (ERSOB)	activada por sobrepaso en la recepción borrable por el usuario

Tabla 14.3 *Registro de banderas*

Ejercicio de comunicación *Full-Duplex*

El programa que se muestra en la **figura 14.40** corresponde a una porción de un ejercicio con la UART en el cual se desarrolla una comunicación *Full-Duplex* (por razones de espacio no se incluyó en su totalidad, pero está disponible en nuestra página de la internet). En este caso se recibe un carácter proveniente de un terminal y de nuevo se retorna el eco hacia el terminal. Este programa sirve para demostrar la interfaz de *software* entre la UART y el programa de aplicación.

El programa empieza enviando los caracteres ASCII de la palabra OK hacia el terminal. Desde ese momento, cualquier carácter recibido del terminal es devuelto como un eco hacia éste. Si por alguna razón en el carácter recibido se detecta un error en su estructura, el microcontrolador retorna el código ASCII SUB (1A) en lugar del carácter recibido. Con ello, el terminal puede saber que hubo un error y puede tomar medidas al respecto, como por ejemplo, enviar de nuevo el dato. Si ha ocurrido un error de sobrepaso, los caracteres recibidos se devuelven pero junto con el carácter ASCII NAK (15). De esta manera, el terminal estaría en capacidad de determinar qué caracteres se perdieron y proceder a enviar los que se detectaron.

Comunicaciones seriales sincrónicas

Como se dijo al principio de esta lección, una comunicación serial sincrónica se caracteriza por transmitir múltiples *bits* de datos, donde cada *bit* transmitido se asocia con un pulso de reloj transmitido por una línea separada de la línea de datos. La comunicación sincrónica puede o no tener *bits* de estructura y ellos pueden o no, estar sincronizados con el reloj.

La transmisión serial de datos distribuye los *bits* en el tiempo y el receptor debe estar en capacidad de reconocer el momento en que empieza y el momento en que termina. Cuando se tiene a disposición una señal de reloj que se pueda utilizar como referencia para ubicar de manera precisa los *bits* de datos, ellos pueden tener diversas duraciones, lo importante es que la línea de reloj delimita el *bit* exacto.

En la **figura 14.41** se muestra un esquema de transmisión sincrónica en donde los ciclos de reloj juegan un papel fundamental. Los *bits* de datos empiezan a ser válidos cada que ocurre un flanco de subida de la señal de reloj y son leídos en el receptor cada que ocurre un flanco de bajada de la misma señal. Podemos notar que la señal de reloj puede ser simétrica o asimétrica.

```
;==================com.grp============
; MANEJO DE LA  UART
;--------------------

;=============================================================================
;MODULO DE TRANSMISION
;=============================================================================

MMTX BTFSS PIR1,TXIF            ; BUFFER DE TX VACIO?
     GOTO MMTX
     MOVF CATUS,W               ; SI -> TOMA EL CARACTER A TRANSMITIR
     MOVWF REGTX;
     RETURN
;-----------------------------------------
; MODULO DE RECEPCION
;-----------------------------------------
RX   CLRF BANREC                ; BORRA BANDERAS DE RECEPCION
     BSF  BANREC,NUBYTE         ; ACTIVAR BANDERA DE BYTE RECIBIDO
     BTFSC ESTREC,FERR          ; ERROR EN LA ESTRUCTURA?
     BSF  BANREC,ERREST1        ; SI -> ACTIVAR BANDERA DE ERROR EN EST. 1
     MOVF REGREC,W              ; TOMAR CARACTER RECIBIDO
     MOVWF REG1                 ; Y LLEVARLO AL BYTE 1
     BTFSS PIR1,RCIF            ; EL DATO AUN ESTA (BYTE 2)EN EL BUFFER?
SALTA GOTO RXX                  ;
     BSF  BANREC,DOSBYTE        ; SI. ACTIVAR LA BANDERA DE RECEPCION DEL BYTE 2
     BTFSC ESTREC,FERR          ; ERROR EN LA ESTRUCTURA?
     BSF  BANREC,ERREST2        ; SI -> ACTIVAR BANDERA DE ERROR 2
     MOVF REGREC,W              ; TOMAR EL 2 CARACTER RECIBIDO
     MOVWF REG2                 ; Y SE MUEVE AL BYTE 2 DEL REGISTRO DEL USUARIO

RXX  BTFSC ESTREC,OERR          ; ERROR POR SOBREPASO
     BSF  BANREC,ERSOB          ; SI -> ACTIVAR BANDERA DE SOBREPASO
     BCF  ESTREC,CREN           ; BORRAR BANDERA DE SOBREPASO DE LA UART
     RETURN
;-----------------------------------------
; SERVICIO DE INTERRUPCION
;-----------------------------------------
SERV_INT
     BTFSS PIR1,RCIF            ; ESTAN LOS DATOS EN EL BUFFER DE RECEPCION?
     GOTO SALINT                ; NO -> NO RECIBIO INTERRUPCION, SALIR
     MOVWF TEMPW                ; SI GUARDAR COPIA DE SEGURIDAD DEL CONTEXTO
     SWAPF ESTADO,W             ;
     BCF  ESTADO,RP0
     MOVWF TEMPST
     CALL RX                    ; TOMAR LOS DATOS RECIBIDOS
SERV_INT1
     SWAPF TEMPST,W             ; RESTAURAR CONTEXTO
     MOVWF ESTADO
     SWAPF TEMPW,F
     SWAPF TEMPW,W

SALINT    RETFIE
;-----------------------------------------
```

Figura 14.39 *Código fuente de la librería para comunicación no sincrónica utilizando la UART*

```
;=========================================
; EJERCICIO DE COMUNICACIONES FULL DUPLEX CON UART
;=========================================
     RADIX HEX
     LIST  P=16F873
optreg     EQU     01
trisa      EQU     05
trisb      EQU     06
trisc      EQU     07
pie1       EQU     0c
pie2       EQU     0d
pcon       EQU     0e
pr2        EQU     12
sspadd     EQU     13
sspstat    EQU     14
ESTTX      EQU     18 ; ESTADO DE LA TX
spbrg      EQU     19
rbif       EQU     0
;-----------------------------------------
TEMPST     EQU     0020
TEMPW      EQU     0021
BANREC     EQU     0022; BANDERA DE RX
REG1       EQU     0023; REG 1 DE LA RX
REG2       EQU     0024; REG 2 DE LA RX
CATUS      EQU     0026; CAR. DE TX DE USUARIO
NUBYTE     EQU     0 ; BANDERA DE BYTE RECIBIDO
DOSBYTE    EQU     1 ; BAN. DE RECEPCION DE 2 BYTES
threbyt    EQU     2
ERREST1    EQU     3
ERREST2    EQU     4
ERSOB      EQU     6  ; ERROR POR SOBREPASO
OH         EQU     04f
KAY        EQU     04b
SUB        EQU     01a ; CARACTER ASCII
NAK        EQU     015
;-----------------------------------------
        org     0x00
INICIO  GOTO    INI
        dw      0072
        dw      006c
        dw      0073
        GOTO    SERV_INT
INI     CLRF    PORTA
        CLRF    PORTB
        CLRF    PORTC
        BSF     ESTADO,RP0
        MOVLW   0c0
        MOVWF   trisc
        MOVLW   000
        MOVWF   trisb
        MOVLW   001
        MOVWF   trisa
        MOVLW   080
        MOVWF   ESTTX
        MOVLW   019
        MOVWF   spbrg
        BCF     ESTADO,RP0
        MOVLW   010
        MOVWF   ESTREC
        BSF     ESTREC,spen
        BSF     ESTADO,RP0
        BSF     ESTTX,TXEN
        BSF     pie1,RCIE
        BCF     ESTADO,RP0
        BSF     INTCON,peie
        CLRF    BANREC
        BSF     INTCON,gie
        MOVLW   OH
        MOVWF   CATUS
        CALL    MMTX
        MOVLW   KAY
        MOVWF   CATUS
        CALL    MMTX
LAZO    BTFSS   BANREC,NUBYTE
        GOTO    LAZO
        BCF     BANREC,NUBYTE
        BTFSS   BANREC,ERREST1
        GOTO    LAZO1
        MOVLW   SUB
        MOVWF   REG1
LAZO1   MOVF    REG1,w
        MOVWF   CATUS
        CALL    MMTX
        BTFSS   BANREC,DOSBYTE
        GOTO    LAZO
        BTFSS   BANREC,ERREST2
        GOTO    LAZO2
        MOVLW   SUB
        MOVWF   REG2
LAZO2   MOVF    REG2,w
        MOVWF   CATUS
        CALL    MMTX
        BTFSS   BANREC,ERSOB
        GOTO    LAZO
        MOVLW   NAK
        MOVWF   CATUS
        CALL    MMTX
        GOTO    LAZO
;-----------------------------------------
        INCLUDE         COM.GRP
;-----------------------------------------
        END
;-----------------------------------------
```

Figura 14.40 *Código ejemplo de comunicación Full-Duplex*

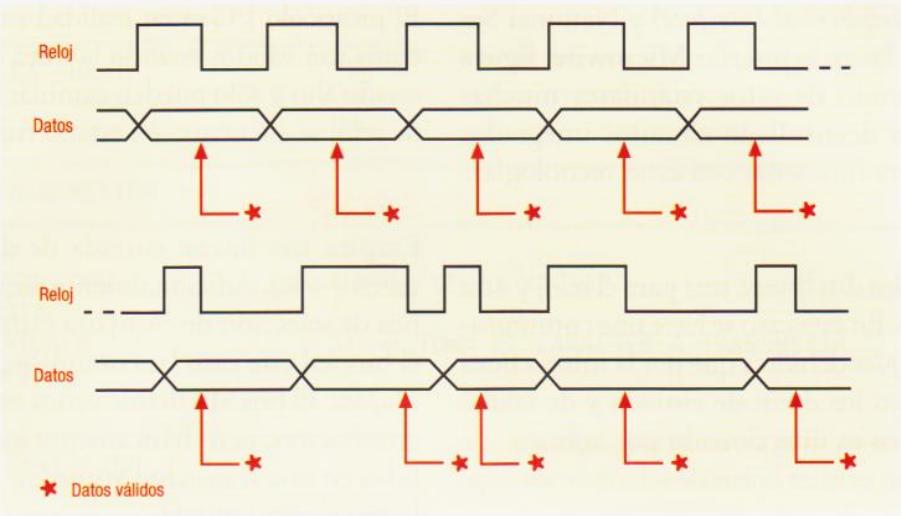


Figura 14.41 *Protocolo serial sincrónico*

Cuando los *bytes* poseen una longitud fija, el receptor solo necesita contar *bits* empezando con el primero que recibe con el fin de delimitarlos. Este tipo de transmisión es utilizada por un protocolo o bus denominado SPI.

En algunos casos se utilizan *bits* de acompañamiento que conforman una estructura que delimita los datos, como en los buses I²C y Microwire.

El bus I²C es muy utilizado para conectar al microcontrolador diversos periféricos que complementen el sistema y poder cumplir con tareas más exigentes. Este bus permite incluir dentro de una misma tarjeta un gran número de dispositivos seriales sin que el *hardware* deba complicarse demasiado y empleando pocas líneas para la conexión.

Protocolo serial sincrónico

El comportamiento de todos los dispositivos que se conectan en el bus serial se rige por las normas dictadas por el protocolo del bus. Como se dijo anteriormente, un protocolo es un conjunto de normas y procedimientos necesarios para asegurar transferencias de datos exitosas. Este define el formato de los *bits* de datos, la forma de la señal de reloj y las medidas de seguridad ante posibles errores en la transmisión o pérdidas en la información.

Dentro del bus sincrónico solo puede existir un dispositivo que gobierne el funcionamiento y control de las comunicaciones en cada instante de tiempo; a este dispositivo se le denomina el **maestro**, los demás dispositivos se les llama **esclavos**. El maestro tiene como tareas iniciar y terminar la transmisión, generar las señales de reloj y direccionar (elegir) el dispositivo esclavo con el que se va a comunicar.

Dispositivos periféricos comunes en la comunicación serial sincrónica

Por lo general los dispositivos seriales desarrollan una tarea periférica específica; los más comunes son:

- Dispositivos de memoria (RAM, EPROM)
- Conversores A/D
- Conversores D/A
- Multiplexores
- Relojes de tiempo real y calendarios
- Puertos I/O
- Generadores de tono (DTMF y tonos musicales)
- Manejadores de *displays*
- Funciones específicas (por lo general I²C)

Estándares de la comunicación serial sincrónica

Dentro de la transmisión serial sincrónica se han establecido tres esquemas muy representativos: Philips desarrolló el **I²C** (*Inter IC*), Motorola el

SPI (*Serial Peripherical Interface*) y National Semiconductor lanzó la interfaz **Microwire**, **figura 14.42**. Partiendo de estos estándares muchas empresas han desarrollado circuitos integrados adaptados para funcionar con estas tecnologías.

El bus I²C

Este bus emplea dos líneas: una para el reloj y otra para los datos. En este caso se hace una comunicación *Half-Duplex* debido a que por la misma línea (SDA) circulan los datos de entrada y de salida. Cada periférico es direccionado por *software* de manera que no existen líneas de selección de *chip*.

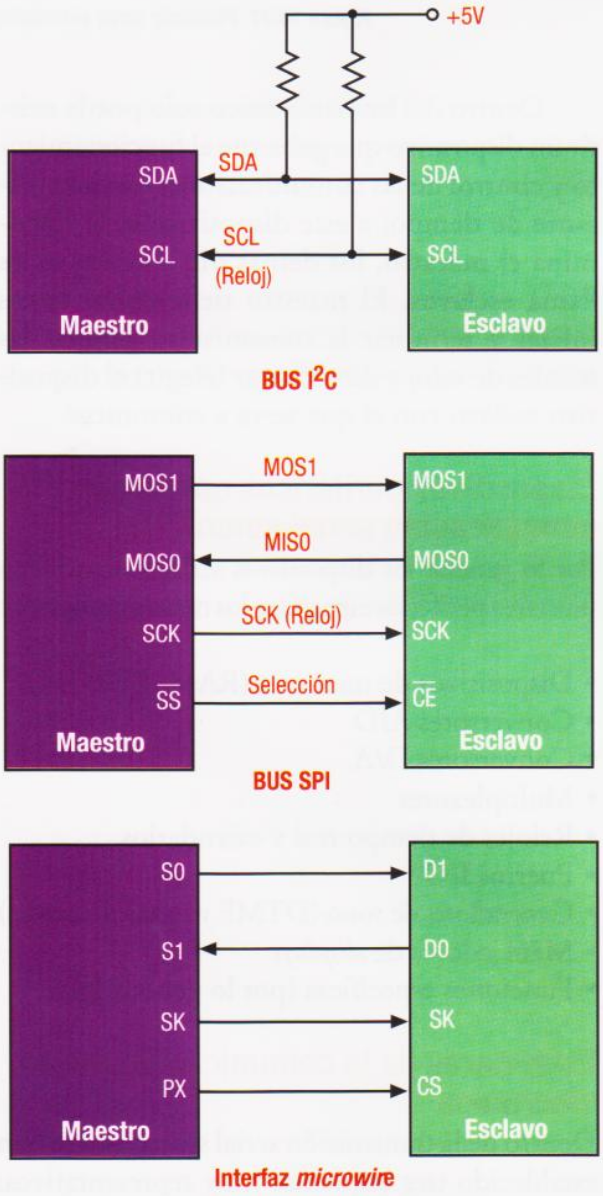


Figura 14.42 Esquema de comunicación serial sincrónica

El protocolo I²C es en realidad muy sencillo; los datos son válidos cuando la línea de reloj está en estado alto y sólo pueden cambiar cuando la línea de reloj se encuentra en estado bajo.

Bus SPI

Emplea tres líneas: entrada de datos, salida de datos y reloj. Adicionalmente requiere de una línea de selección de *chip* para cada dispositivo en el bus; en este caso las comunicaciones son *Full-Duplex*. El bus SPI define varios esquemas de sincronización, pero básicamente los datos son válidos en una transición y pueden modificarse en la transición opuesta.

Interfaz Microwire

Es similar al bus SPI: requiere tres líneas, más la línea de selección de *chip*. En este caso los datos son válidos durante la transición de estado bajo a estado alto.

Profundizemos un poco en el bus I²C, el cual es ampliamente utilizado en todo el mundo y basándonos en él desarrollaremos una serie de ejercicios en la sección de programación de este curso.

El bus I²C

I²C (*Inter-Integrated Circuit*) es un bus serial de dos hilos utilizado para transferencias de datos en 8 *bits*, **figura 14.43**. Los dos hilos (reloj serial y datos seriales) transportan la información entre los dispositivos conectados al bus, los cuales se clasifican entre maestros y esclavos. La línea de datos es bidireccional pero la información solo puede fluir en una sola dirección en un tiempo dado.

En un bus I²C todos los dispositivos pueden funcionar como transmisor o como receptor, pero no simultáneamente; sin embargo, algunos dispositivos sólo sirven como receptores y otros como transmisores; por ejemplo, un *display* es un dispositivo que sólo puede ser receptor.

Dentro del bus I²C el dispositivo maestro es el encargado de iniciar la transferencia de información, generar la señal de reloj y lle-

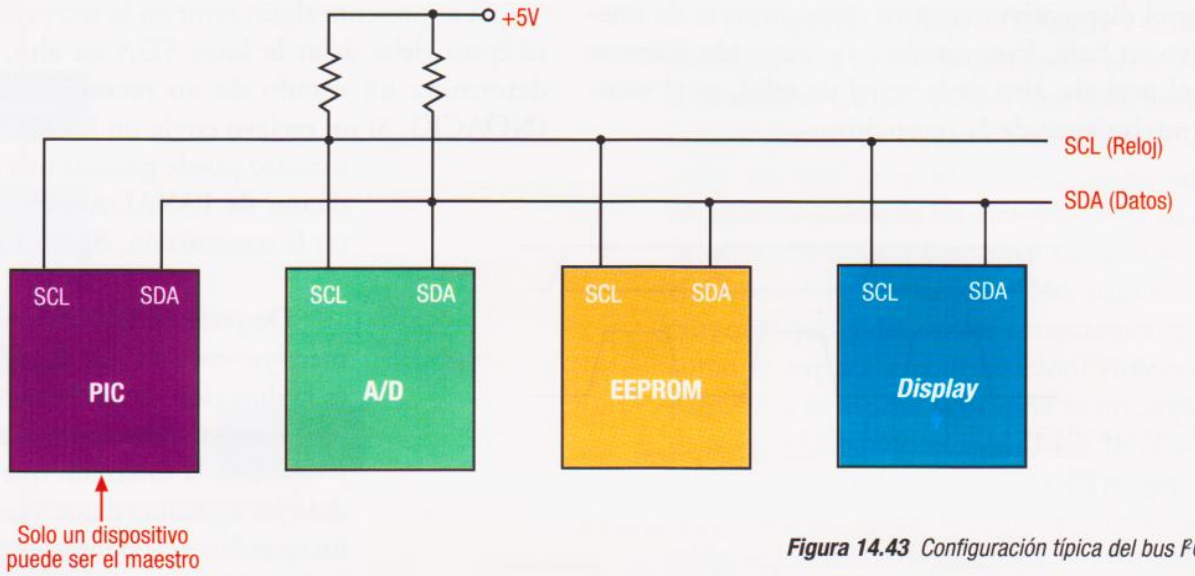


Figura 14.43 *Configuración típica del bus I²C*

var el control de las operaciones. Para facilitar esta tarea, cada dispositivo dentro del bus posee una dirección única por medio de la cual el maestro puede entablar una comunicación precisa con ellos. Es de resaltar que en el mismo bus pueden existir varios dispositivos maestros, pero no al mismo tiempo. Para evitar que se presenten colisiones por este concepto, se debe poner en claro un procedimiento de arbitramento.

Especificaciones del bus I²C

Las dos líneas del bus I²C: la de reloj (SCL) y la de datos (SDA) son bidireccionales y se conectan a una fuente de voltaje positiva a través de resistencias de *pull-up*, de manera que cuando el bus está libre, ambas líneas se encuentran en estado alto. Las velocidades que maneja el bus I²C se encuentran en el rango de 0 hasta 100 Kbits/segundo.

La generación de las señales de control es responsabilidad del maestro. Él genera la señal de reloj cuando la información es enviada hacia el bus, y cuando hay comunicación entre dos dispositivos, se debe establecer una serie de condiciones de reconocimiento entre ambos elementos, pero igualmente sincronizadas con el reloj. Para ello se utiliza un bit de reconocimiento denominado ACK (*acknowledgment*).

El inicio y la parada del proceso son señales de control generadas por el maestro; cada transferencia de información debe presentarse primero con una condición de INICIO, seguida de la dirección del esclavo, los *bytes* de datos y finalmente una condición de PARADA.

La condición de INICIO se define como una transición de alto a bajo en la línea SDA, mientras la línea SCL se encuentra en alto; y una condición de PARADA se define como una transición de estado bajo a estado alto en la línea SDA, mientras la línea SCL se encuentra en alto, **figura 14.44**.

Cada *byte* que se ponga en la línea SDA debe ser de 8 *bits*, pero se pueden transmitir todos los *bytes* que se desee. Cada uno de ellos debe estar acompañado del *bit* de reconocimiento (ACK) generado en el receptor, el cual indica que el receptor pudo leer y procesar el dato correctamente, y cada uno se transmite empezando por el *bit* MSB. Los datos son válidos durante el periodo alto del reloj y pueden cambiar durante el periodo bajo del reloj.

La señal de ACK necesita de la señal de reloj enviada desde el maestro. Para conformar esta señal, una vez terminados los datos, el transmisor debe poner en alto la línea SDA

y el dispositivo receptor debe ponerla de nuevo en bajo. Este estado bajo ocurrido durante el periodo alto de la señal de reloj, es el reconocimiento de la recepción.

Si se presenta algún error en la recepción, el receptor debe dejar la línea SDA en alto, esto determina un estado de no reconocimiento (NOACK). Si un esclavo envía un NOACK, el maestro puede generar una condición de PARADA para abortar la transmisión, **figura 14.45**.

Después del INICIO, el primer *byte* enviado por el maestro es la dirección del esclavo con el cual desea entablar comunicación y establece la dirección que tendrán los siguientes datos. Para ello los siete *bits* más significativos de la dirección del esclavo indican la dirección del dispositivo y el *bit* menos significativo (LSB) determina el flujo de la información. Un cero en el *bit* LSB quiere decir que el maestro transmitirá en los siguientes *bytes* y un uno en el mismo *bit* indica que transmitirá el esclavo, **figura 14.46**.

Observemos en la **figura 14.47** una gráfica donde se muestra una transmisión I²C donde el dispositivo esclavo es una memoria EEPROM. La señal ACK generada por el esclavo es lo que sigue después de la dirección del esclavo y a partir de ella deben estar los *bytes* de datos; luego va la señal ACK generada por el receptor.

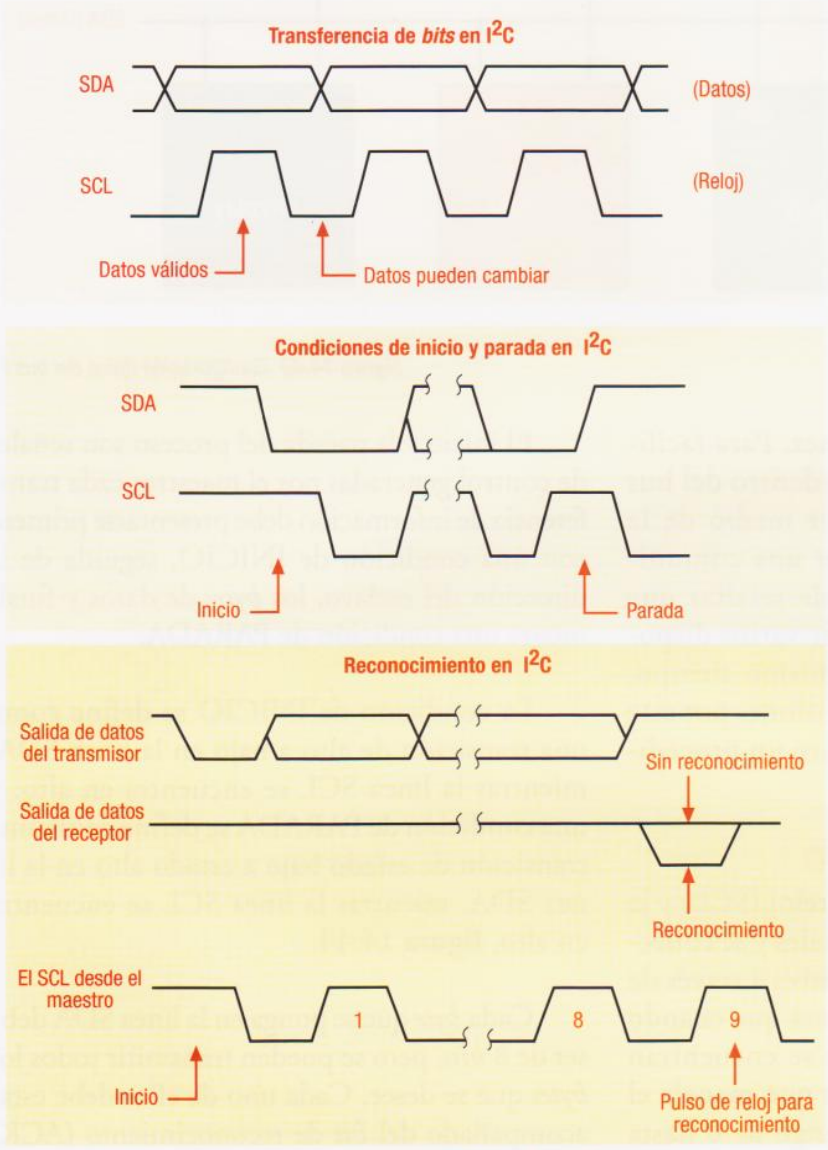


Figura 14.44 *Funcionamiento del bus I²C*

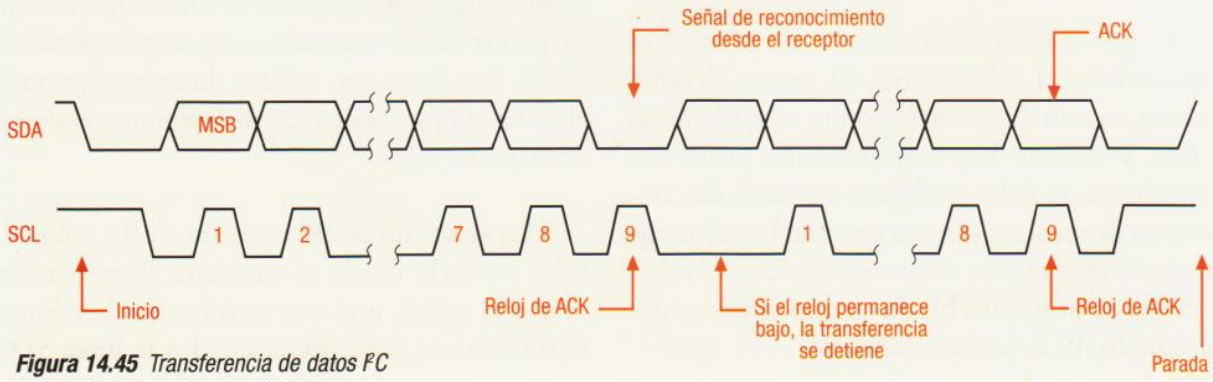


Figura 14.45 *Transferencia de datos I²C*

Si en un momento dado se desea cambiar la dirección de los datos, se debe enviar de nuevo una condición de INICIO y la dirección del esclavo para establecer el cambio. Finalmente la comunicación con el esclavo se termina con el envío de una señal de PARADA.

Figura 14.46 *Formato de la dirección del esclavo*

El esquema mostrado en la **figura 14.47** corresponde a un ejemplo concreto de una comunicación por medio del bus I²C, entablada con una memoria EEPROM, la cual es un dispositivo que puede enviar o recibir datos. De la misma manera este esquema se cumple para otros dispositivos como visualizadores, decodificadores, controladores, etc. En la sección de programación analizaremos un ejemplo en el que, a través del bus I²C, incorporamos una memoria serial a un sistema microcontrolado.

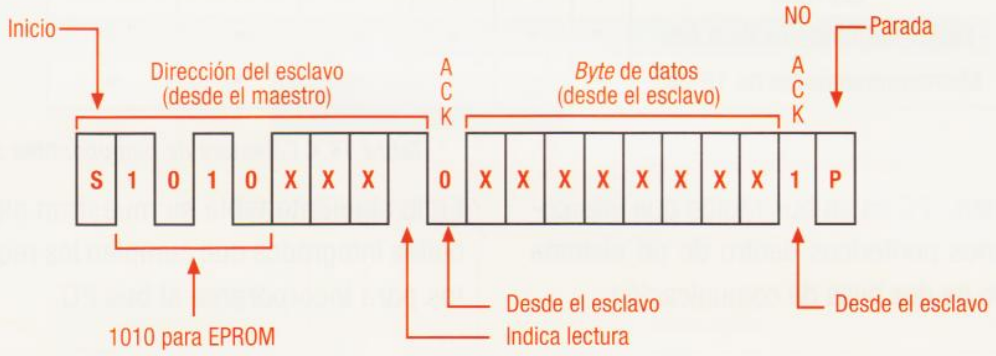


Figura 14.47 *Transmisión I²C*

PARA SABER MÁS

Breve historia del I²C

El bus I²C se desarrolló en los años 80 e inmediatamente fue clasificado dentro de la categoría de redes LAN. En la actualidad millones de televisores, radios y otros aparatos utilizan este sistema de comunicación en sus circuitos internos; esta adopción se debió al alto interés que despertaron sus ventajas dentro del mundo de la microelectrónica. Igual acogida tuvo este protocolo de comunicación en la electrónica automotor para comunicar la mayoría de los equipos instalados en los automóviles. El alto impacto del bus I²C motivó a muchos fabricantes a adoptar esta tecnología en sus productos, logrando una reducción en costos y aumentando los volúmenes de producción.

De esta manera surgieron numerosas gamas de productos (por ejemplo, convertidores A/D, D/A, memorias RAM, etc.) y entró al mercado de la telefonía moderna, donde son indispensables elementos tales como las memorias, los generadores de frecuencias, las pantallas de cristal líquido, etc. Ahora la mira está puesta en campos como la multimedia. En la **tabla 14.4** se muestran algunas de las posibilidades de cobertura en mercados de componentes del bus I²C.

	TV	Radio	Audio	Teléfono	Industria	Automóvil
Recepción de TV	•					
Recepción de radio		•				
Tratamiento de audio	•	•	•	•	•	•
Mando infrarrojo	•	•	•		•	•
Mando de *displays* LCD	•	•	•	•	•	•
Mando de *displays* LED	•	•	•	•	•	•
Sintetizador de voz					•	•
DTMF			•	•	•	
I/O	•			•	•	•
Convertidor D/A y A/D	•	•	•	•	•	•
Reloj/Temporizador	•	•	•	(•)	•	•
RAM	•	•	•	•	•	•
EEPROM	•	•	•	•	•	•
Microcontroladores de 8 *bits*	•	•	•	•	•	•
Microcontroladores de 16 *bits*	•	•	•	•	•	•

Tabla 14.4 *Cobertura de componentes con bus I²C*

En resumen, I²C es un bus rápido que interconecta varios periféricos dentro de un sistema por medio de dos hilos de comunicación.

En la siguiente tabla se muestran algunos circuitos integrados que cumplen los requerimientos para incorporarse al bus I²C.

Integrado	Función	Dirección						
		A6	A5	A4	A3	A2	A1	A0
PCD3311/12	Generador de tonos DTMF/modem/musicales	0	1	0	0	1	0	U
PCF8200	Sintetizador de voz	0	0	1	0	0	0	0
PCF8570	RAM 256*8	1	0	1	0	U	U	U
PCF8573	Reloj/Calendario	1	1	0	1	0	U	U
PCF8574	Convertidor de I²C a 8 *bits*	0	1	0	0	U	U	U
PCF8579	Manejador de LCD	0	1	1	1	1	0	U
PCF8581	EEPROM de 128 *bytes*	1	0	1	0	U	U	U
PCF8591	Convertidor de 8 *bits* A/D y D/A	1	0	0	1	U	U	U
SAA7199	*Encoder* digital	1	0	1	1	0	0	0
SAA1300	Manejador de corriente	0	1	0	0	0	U	U
TDA4680	Procesador de vídeo	1	0	0	0	1	0	0
TDA8421	Procesador de audio estéreo HI-FI	1	0	0	0	0	0	U
TDA8461	Decodificador de color	1	0	0	0	1	0	U
TEA6100	Interfaz de sintonización	1	1	0	0	0	0	1
TSA5511	Sintetizador de frecuencia PLL para TV	1	1	0	0	0	U	U
TSA6057	Sintetizador de frecuencia PLL para radio	1	1	0	0	0	U	U

U: Usuario

LECCIÓN 15

LOS CONVERTIDORES ANÁLOGO/DIGITALES

Estos módulos son muy importantes para la gran mayoría de diseños en instrumentación y control de procesos industriales, debido a que permiten leer variables análogas de sistemas reales y registrarlas en la memoria. En esta lección analizaremos el funcionamiento y el manejo de este módulo interno.

En los microcontroladores más modernos de Microchip tenemos la posibilidad de leer señales análogas por medio de un módulo interno especializado. Sus principales características son:

- Entre 5 y 8 canales de entrada (dependiendo de la referencia del microcontrolador)
- Un multiplexor análogo
- Un circuito de ajuste y retención de la señal de entrada
- Varias alternativas de fuentes de reloj para llevar a cabo la conversión
- Velocidad de toma de muestras (muestreo) ajustable
- Selección de un voltaje de referencia interno o externo
- Conversión a 8 ó a 10 *bits* (dependiendo de la referencia)
- Generación de interrupciones cuando se ha completado la conversión

que internamente se haga esta conexión y no tengamos que emplear un pin para aplicar un voltaje de referencia externo; de esta manera se logra que los PIC puedan utilizar hasta 8 canales de entrada A/D.

El caso más apropiado para hacer corresponder el voltaje de referencia con el voltaje de la alimentación del PIC, es cuando se trabaja con un transductor cuya salida es proporcional a su propia fuente de alimentación, permitiéndose un amplio rango de voltajes análogos de entrada, entre 0V y el voltaje de referencia:

$$0V \leq V_{ENTRADA} \leq V_{REFERENCIA}$$

Para configurar el funcionamiento de los convertidores A/D debemos seguir el procedimiento

Características del convertidor AD

Analicemos el caso de la conversión a 8 *bits*. El convertidor análogo/digital maneja la función de transferencia mostrada en la **figura 15.1**, en donde se convierte el voltaje de entrada en un número de 8 *bits*.

La señal análoga de entrada al convertidor se acondiciona mediante un factor de escala denominado voltaje de referencia, produciendo una salida en 8 *bits* como se muestra en dicha figura.

Una de las opciones más utilizadas para configurar el voltaje de referencia para el convertidor, es que éste corresponda al voltaje de alimentación del PIC. Para hacer uso de esta opción, debemos hacer algunas configuraciones dentro del programa para

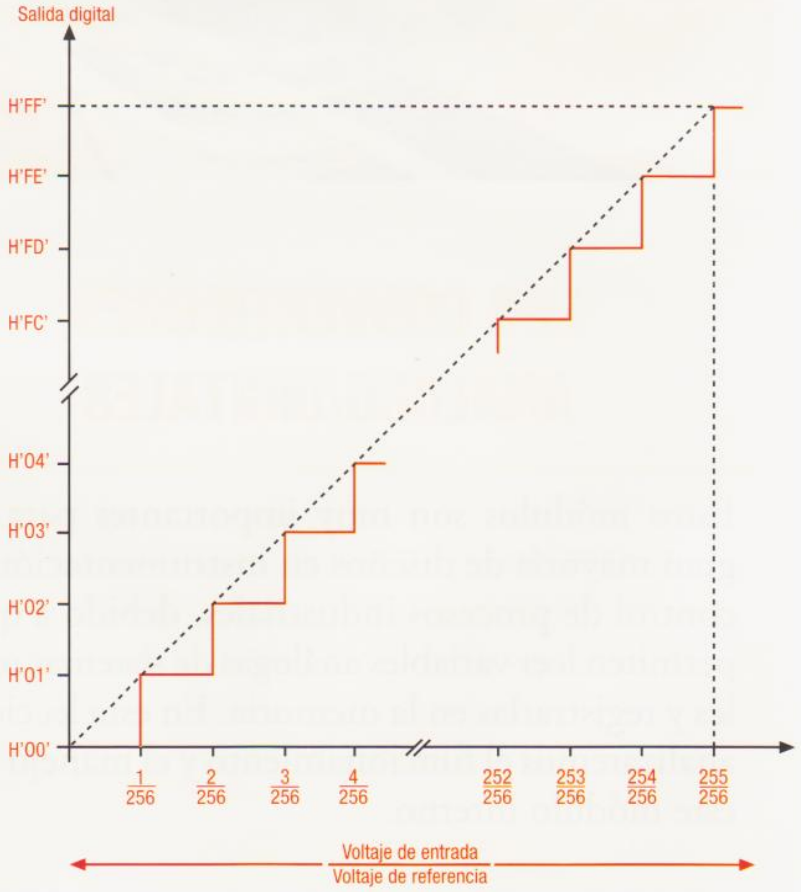
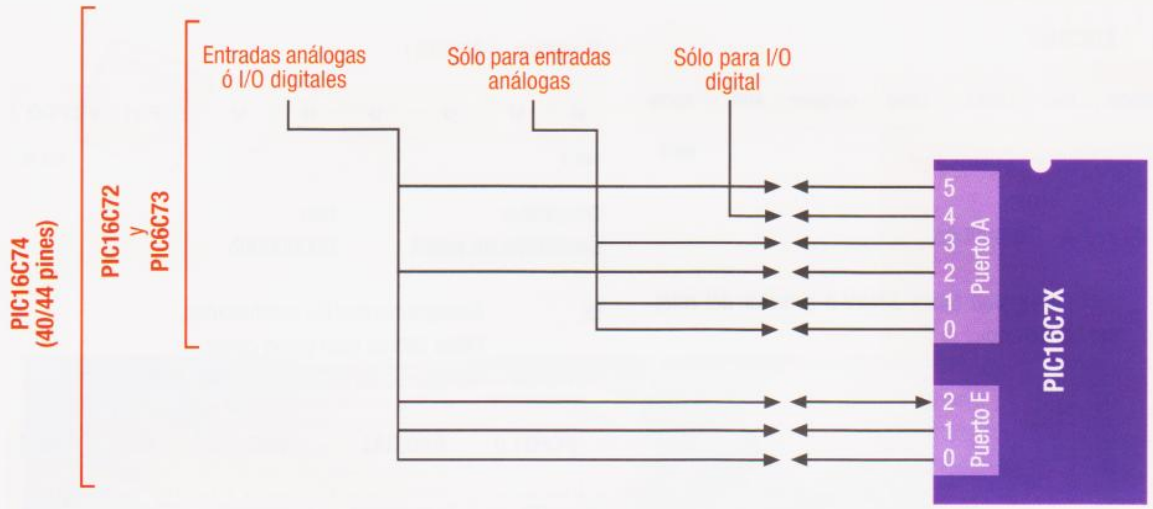


Figura 15.1 *Función de transferencia del conversor A/D de los PIC*



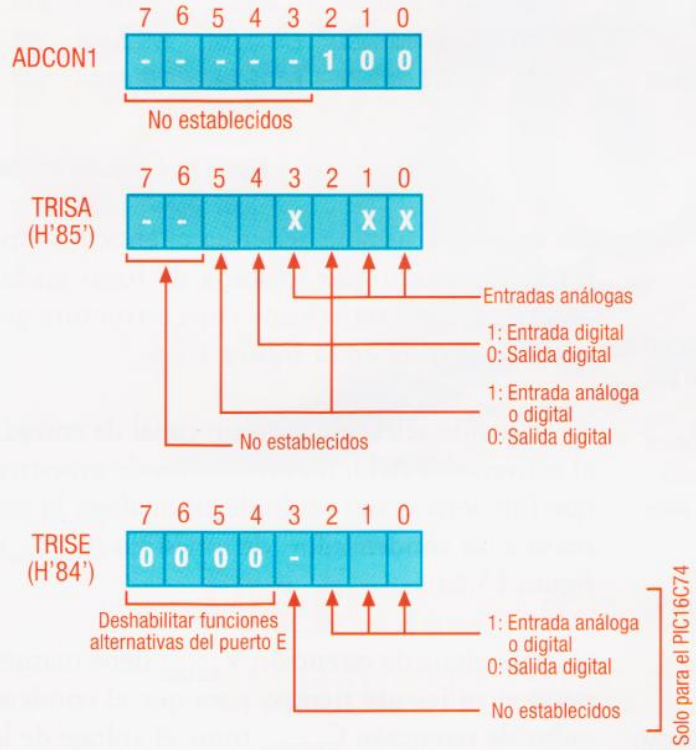


Figura 15.2 *Entradas análogas con* $V_{REF} = V_{DD}$

mostrado en la **figura 15.2**. En esta figura se muestran inicialmente los pines con funciones A/D y la configuración de los registros asociados. Cualquier pin al que le sea asignada la función de entrada análoga o digital puede leerse como una entrada digital o como una entrada análoga.

El módulo A/D de 8 *bits* tiene tres registros asociados, uno para almacenar el resultado (ADRES) y dos de control (ADCON0 y ADCON1), **figuras 15.3** y **15.4**; el módulo A/D de 10 *bits* tiene cuatro registros asociados, dos para almacenar el dato (ADRESH y ADRESL) y dos de control (ADCON0 y ADCON1).

En algunos casos es más aconsejable utilizar una referencia de voltaje externa de 3,0 voltios para proporcionar una mayor resolución en la salida (para mediciones de voltajes análogos por debajo de 3,0 voltios). Esto es especialmente llamativo para mediciones de voltajes diferenciales.

La **figura 15.5** muestra la conexión de una referencia de voltaje externa al PIC y adicionalmente, la configuración de los registros para su funcionamiento.

Características de funcionamiento del conversor A/D

Resumamos las principales características de funcionamiento de los convertidores A/D internos de los microcontroladores PIC.

Voltaje de referencia. **Interno:** $V_{REFERENCIA} = V_{DD}$ **(voltaje de alimentación) Externo:** $3{,}0V \leq V_{REFERENCIA} \leq V_{DD} + 0{,}3\,V$.

Corriente empleada por el A/D. Típicamente 180 uA.

Registro: ADCON0

ADSC1	ADSC0	Res.	CHS1	CHS0	Go/Done	ADIF	ADON
bit 7							bit 0

Dirección: 08h
condición de *reset*: 00000000b

ADSC1,0: *A/D Conversion Clock Select* ó selector del reloj del convertidor

00	fosc/2
01	fosc/8
10	fosc/32
11	frc

Res: Reservado
Este *bit* no se utiliza

CHS1,0: *Analog Channel Select* ó selector del canal análogo

00	canal 0 (AIN0)
01	canal 1 (AIN1)
10	canal 2 (AIN2)
11	canal 3 (AIN3)

GO/DONE: Comienza la conversión cuando se coloca en 1. Al finalizar la conversión es puesto en 0 por *hardware*.

ADIF: *A/D Conversion Complete Interrupt Flag* ó bandera de interrupción por finalización de la conversión. Se coloca en 1 cuando termina la conversión; debe ser puesta en 0 por programa.

ADON: *Bit* para controlar la activación del convertidor.
0: módulo del convertidor desconectado y por lo tanto no exige corriente.
1: módulo del convertidor A/D en operación.

Figura 15.3 *Registro ADCON0*

Tiempo de muestreo mínimo. 12 us. Éste es el tiempo transcurrido desde que el canal se ha seleccionado y antes de que se inicie la conversión.

Tiempo de conversión.

- 15 us para un oscilador de 20 MHz
- 30 us para un oscilador de 10 MHz
- 19 us para un oscilador de 4 MHz

Una de las características más importantes que debemos considerar al utilizar procesos de conversión, es tener en cuenta el tiempo de muestreo que se acaba de listar en las anteriores características y cuya estructura general se aprecia en la **figura 15.6**.

Registro: ADCON1

U	U	U	U	U	U	PCFG1	PCFGO
bit 7							bit 0

Dirección: 88h
Condición de *reset*: 00000000b

U: *Unimplemented* No establecidas.
Estos *bits* se leen como ceros.

PCFG1,0: Configuración de los pines RA0-RA3

PCFG1,0	RA0,RA1	RA2	RA3	Vref
00	E. análogas	E. análoga	E. análoga	Vdd
01	E. análogas	E. análoga	Referencia	RA3
10	E. análogas	E/S digital	E/S digital	Vdd
11	E/S digitales	E/S digital	E/S digital	Vdd

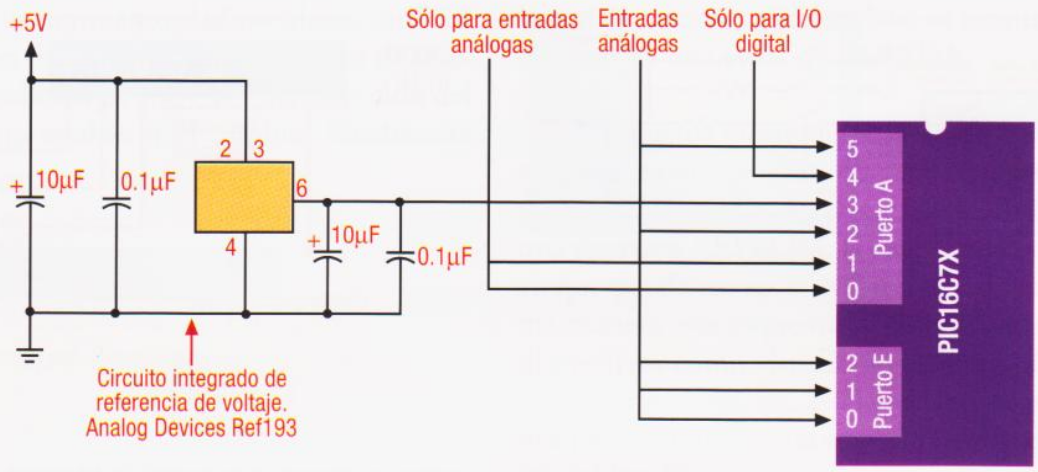
Figura 15.4 *Registro ADCON1*

Cuando seleccionamos un canal de entrada al convertidor A/D, un interruptor de muestreo que funciona como multiplexor análogo lo conecta a un condensador de muestreo (C_{RETEN}), **figura 15.6a**.

El voltaje de retención V_{RETEN} debe mantenerse el suficiente tiempo para que el condensador de retención C_{RETEN} tome el voltaje de la fuente V_{FUENTE} y alcance uno de los escalones de voltaje mostrados en la figura 15.1.

Entre mayor sea la resistencia de la fuente, mayor será el tiempo de carga del condensador, **figura 15.6b**. Una resistencia alta en la fuente se puede reducir mediante un amplificador en configuración de seguidor, **figura 15.6c**.

Después de esperar el tiempo de muestreo, se puede iniciar la conversión. El circuito A/D abrirá el interruptor de muestreo y



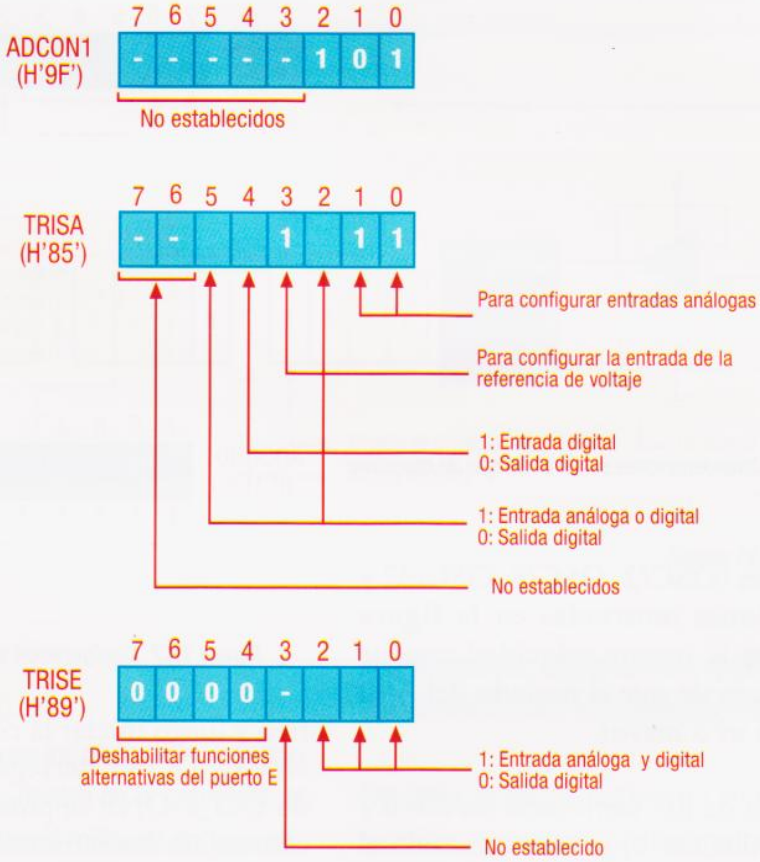


Figura 15.5 *Entradas A/D con voltaje de referencia externo*

llevará a cabo la conversión del voltaje de entrada en el momento en que el interruptor se abre. Si el A/D se utiliza para tomar muestras en un solo canal a iguales intervalos de tiempo, se puede hacer uso de interrupciones.

Utilización del convertidor A/D

Para trabajar con el convertidor interno de los microcontroladores PIC debemos seguir los pasos mostrados en las figuras 15.2 y 15.5, en donde se configuran los registros ADCON1, TRISA y TRISE para seleccionar el voltaje de referencia y los canales de entrada deseados.

Luego, se inicializa el registro ADCON0 mediante los pasos que se muestran en la **figura 15.7**. El primer paso es seleccionar la fuente de reloj del convertidor A/D entre cua-

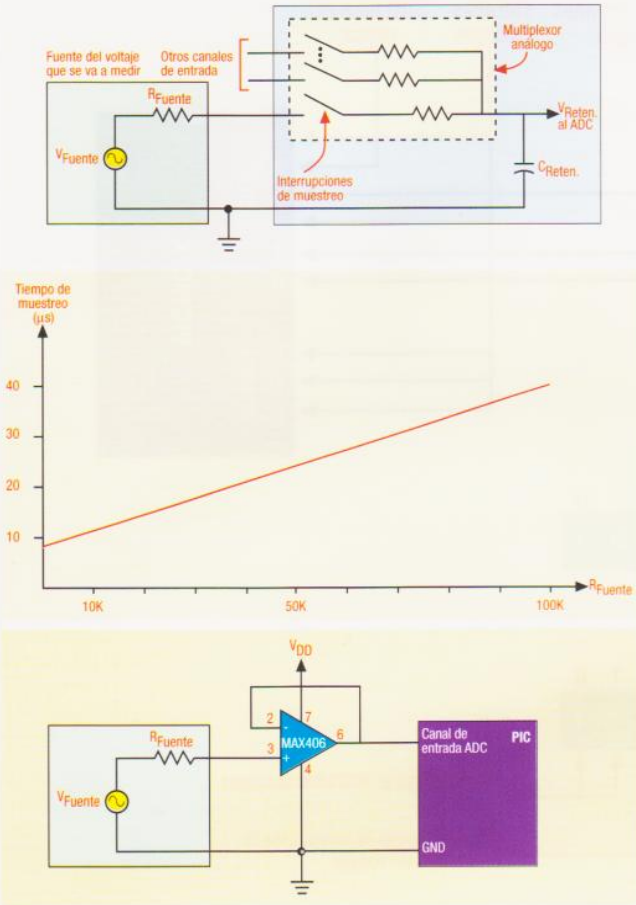


Figura 15.6 *Consideraciones sobre el tiempo de muestreo*

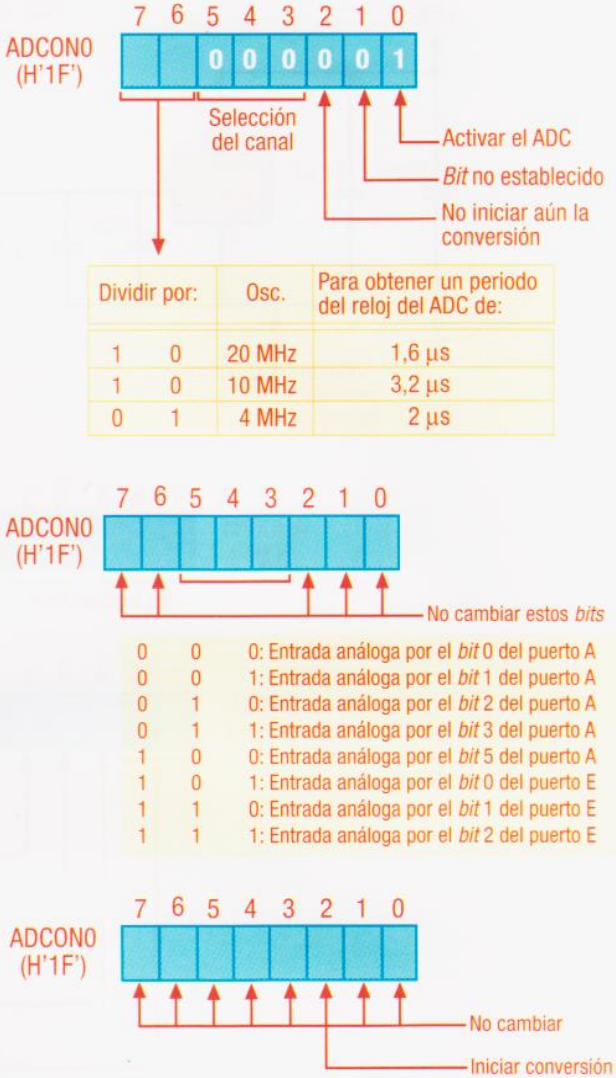


Figura 15.7 *Configuración y utilización del registro ADCON0*

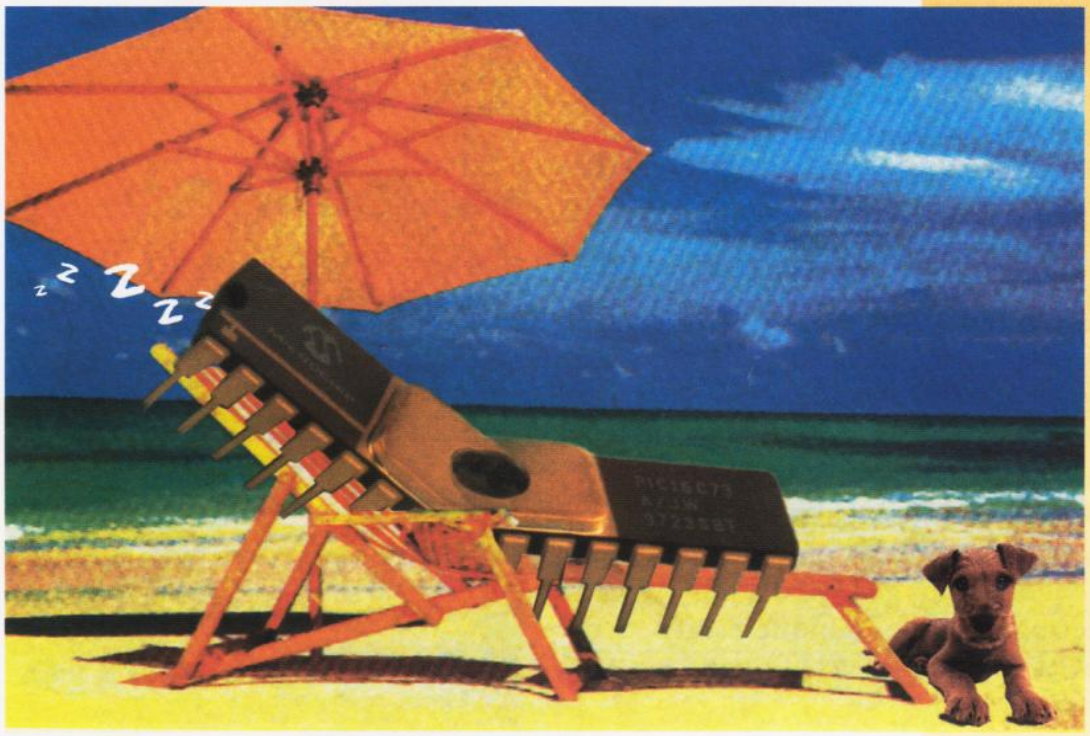
tro posibilidades (OSC/2, OSC/8, OSC/32 y RC). Las opciones mostradas en la **figura 15.7a** permiten la mayor velocidad posible bajo la restricción de que el periodo del A/D debe ser de 1,6 us o mayor.

La selección de RC (activando los *bits* 6 y 7 del registro ADCON0) se utiliza cuando el microcontrolador trabaja con un cristal de frecuencia relativamente baja.

Si sólo vamos a utilizar una entrada análoga, podemos seleccionarla como se ilustra en la **figura 15.7b** y la selección del periodo mostrada en la 15.7a. Si vamos a utilizar varias entradas análogas, es importante esperar un tiempo de muestreo como se apreció en la figura 15.6b. Esto es, una vez seleccionado el canal, debemos esperar un tiempo de muestreo y luego iniciar la conversión activando el *bit* GO_DONE del registro ADCON0. Cuando GO_DONE se pone de nuevo en bajo, la conversión ha terminado, y podemos leer el resultado del registro ADRES (H'1E').

Cuando se presenta la necesidad de tomar muestras periódicamente de uno o más canales de entrada, un excelente recurso es pensar en la posibilidad de incorporar dentro del funcionamiento del convertidor el módulo timer1/CCP2, del cual ya analizamos su operación en las lecciones 12 y 13 de esta sección, para que periódicamente se encargue de iniciar el proceso de conversión.

LECCIÓN 16



EL WATCHDOG Y EL MODO SLEEP

El temporizador **watchdog** y el modo **sleep** son dos conceptos que, si se saben manejar correctamente, pueden traernos enormes beneficios para la estabilidad y optimización de los sistemas electrónicos que desarrollemos con microcontroladores. En esta lección analizaremos estos factores y observaremos como insertarlos correctamente dentro del código del programa.

El temporizador *watchdog* (WDT)

Como su nombre lo sugiere, el **watchdog** o perro guardián es un módulo interno que una vez activado, permanece alerta para que los programas no caigan en fallas inesperadas e indeseables. Fundamentalmente su vigilancia consiste en generar cada cierto tiempo un estado de *reset* para que el programa inicialice nuevamente su operación desde la dirección 0x0000. De esta manera combatimos problemas tales como caer en ciclos infinitos o alteraciones debidas al ruido. Muchos usuarios acostumbran emplear dentro de sus aplicaciones, alguno de los temporizadores para que hagan las funciones del **watchdog**.

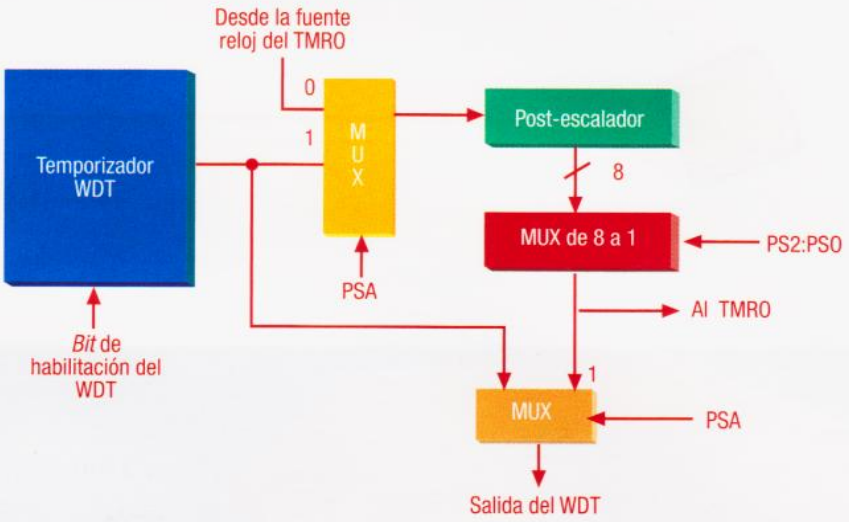


Figura 16.1 *Estructura en bloques del watchdog*

El temporizador **watchdog** (WDT) es un oscilador interno basado en un sencillo circuito RC, cuyo funcionamiento es absolutamente independiente del oscilador general del microcontrolador de manera que, aunque este último se detenga, el **watchdog** (WDT) continuará funcionando.

En la **figura 16.1** podemos observar una estructura en bloques del WDT en donde vemos que el temporizador se activa mediante un *bit* de configuración; si recordamos las lecciones anteriores, en la posición 3 del registro OPCION se encuentra el *bit* PSA, **figura 16.2**. Si este *bit* se encuentra en uno, el prescalador se asigna al WDT y si está en cero, se asigna al TMR0; haciendo la asignación del prescalador al WDT hacemos que los pulsos de entrada al TMR0 no sean sometidos a ningún proceso de división de frecuencia.

De esta manera el WDT puede ser sometido a un proceso de división para variar su tiempo de respuesta, el cual se rige por los mismos factores configurados por los *bits* PS0, PS1 y

Registro: OPCION

RBPU	INTEDG	GRTS	RTE	PSA	PS2	PS1	PS0

***Bit* 7** ***Bit* 0**

Dirección: 81h
condición de *reset*: 11111111b

RBPU: *PortB Pull-up Enable* o Habilitación de *pull-up* del puerto B.
0: habilita las *pull-ups* internas
1: las deshabilita

INTEDG: *INT Interrupt Edge Select* o Selector de flanco de la interrupción INT
0: flanco de bajada
1: flanco de subida

RTS: *TMR0 Signal Source* o Fuente de señal de TMR0.
0: ciclo de instrucciones interno (Temporizador)
1: transición en el pin RA4/TOCK (Contador)

RTE: *TMR0 Signal Edge* o Flanco de la señal TMR0
0: incremento en transición de bajo a alto
1: incremento en transición de alto a bajo

PSA: *Prescaler Assignment* o Asignación del preescalador
0: TRM0 (Contador/Temporizador)

Valor	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Figura 16.2 *Registro OPCION*

PS2. Cuando el prescalador es asignado al WDT se acostumbra denominarlo post-escalador.

Uno de los problemas más comunes que se encuentran cuando depuramos un programa, es que el temporizador **watchdog** (WDT) está habilitado inadvertidamente y el circuito programado no hace su trabajo adecuadamente. El WDT se puede activar mediante la configuración establecida dentro del entorno de desarrollo, **figura 16.3**; también se puede hacer dentro del código del programa por medio de la **palabra de configuración**. Debemos tener mucho cuidado en este punto, porque si dentro del programa desactivamos el **watchdog**, pero en el entorno de programación se activa, este módulo en definitiva quedará habilitado y esto se puede reflejar en problemas de funcionamiento de nuestro dispositivo electrónico.

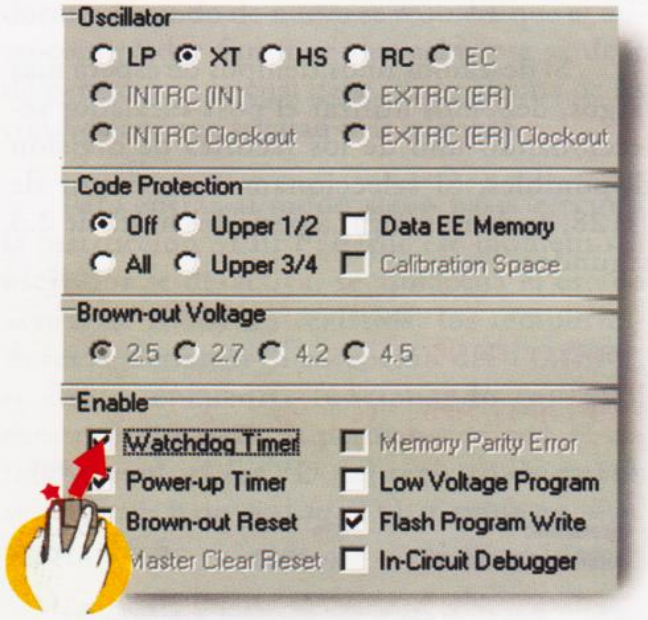


Figura 16.3 *Parámetros de configuración*

La palabra de configuración

Cuando se programa un microcontrolador PIC, dentro del código se suele colocar la palabra de configuración que corresponde a una técnica que nos permite configurar desde el programa los parámetros con que va a funcionar el circuito; para hacer esta operación se puede colocar un número binario o ciertas etiquetas habilitadas, como se muestra en la **figura 16.4**. Un ejemplo de esta configuración la podemos observar en el proyecto **Aviso luminoso con diodos LED** publicado en este curso. Para especificar los parámetros deseados se utiliza la directiva del ensamblador *config*, la cual permite especificar cada uno de estos *bits* dentro del código del programa. Esta directiva hace una operación AND (&) con cada una de las etiquetas de la **figura 16.4**, por ejemplo:

```
_config (_XT_OSC & _WDT_OFF &
_PWRTE_OFF & _BODEN_OFF &
_CP_OFF)
```

En la sintaxis de esta directiva podemos observar algunos caracteres **underline** (_) que se deben escribir obligatoriamente. Como hemos

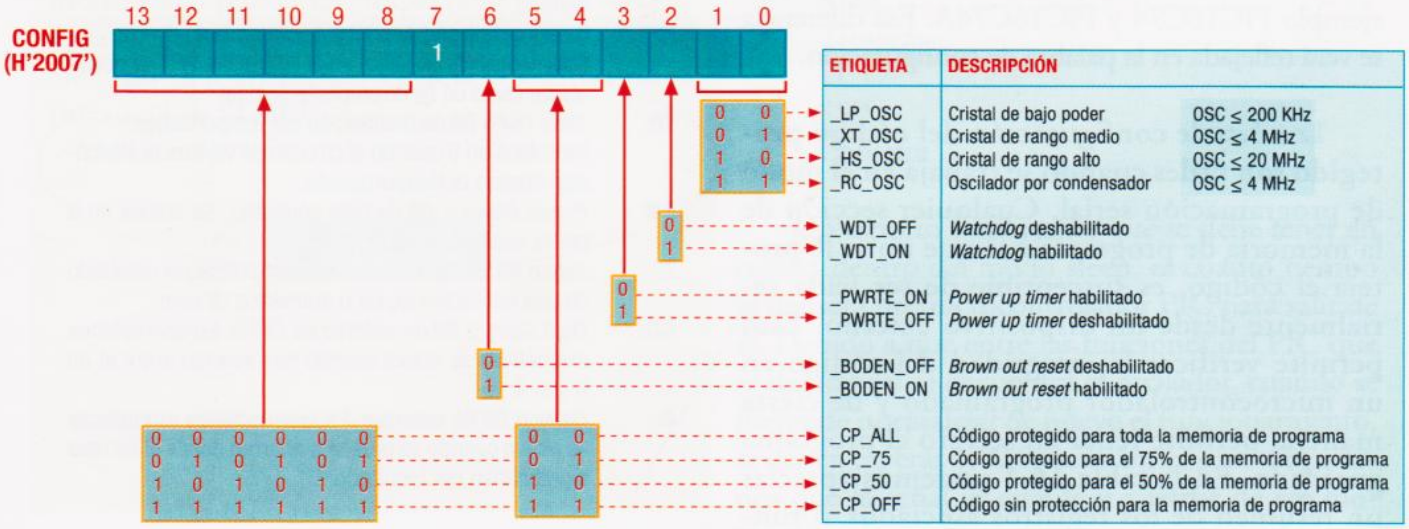


Figura 16.4 *Palabra de configuración*

Nombre	*Bit* 7	*Bit* 6	*Bit* 5	*Bit* 4	*Bit* 3	*Bit* 2	*Bit* 1	*Bit* 0
config.*bits*	MPEEN	BODEN	CP1	CP0	$\overline{PWRTE}$	WDTE	FOSC1	FOSC0
OPTION_REG	$\overline{RBPU}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Tabla 16.1 *Registros asociados al WDT*

mencionado anteriormente, muchos programadores acostumbran incluir dentro de sus programas una serie de archivos que contienen subrutinas, definiciones, librerías, etc., que los hacen más comprensibles, a la vez que permiten reutilizar componentes de *software* en futuras aplicaciones.

El más utilizado es el archivo INC, el cual contiene todo el mapa de direcciones RAM y la descripción de cada uno de los *bits* de los distintos registros de cada tipo de microcontrolador, lo cual nos ahorra el trabajo de repetir continuamente las mismas definiciones; ejemplos de estos archivos son el P16C74A.INC, P16F84.INC, etc. Debido a la gran utilidad de estos archivos, ellos se encuentran disponibles en la página del fabricante para que sean descargados y utilizados libremente. Dentro de estos archivos INC también se encuentran todas las definiciones de la palabra de configuración, lo que nos permite utilizar las etiquetas en el momento que vayamos a programarla.

En muchas ocasiones se encontrará que un mismo microcontrolador aparece con el sufijo A, por ejemplo PIC16C74 y PIC16C74A. Esa diferencia se verá reflejada en la palabra de configuración.

Los *bits* de configuración del **código protegido** son útiles cuando se trabaja en el modo de programación serial. Cualquier sección de la memoria de programa a la que no se le proteja el código, es susceptible de ser leída serialmente desde un dispositivo externo. Esto permite verificar la integridad del código en un microcontrolador programado y de cierta manera, permite tener en secreto el programa guardado. En la **tabla 16.1** podemos apreciar un resumen de los registros asociados al funcionamiento del temporizador **watchdog**.

Operación del watchdog

En condiciones normales, el WDT se encarga de generar un estado de *reset* al microcontrolador cada que se alcanza un tiempo de espera. Cuando se trabaja con el post-escalador, el WDT tiene un periodo de espera de 18 ms el cual puede variar con parámetros externos como la temperatura.

Si deseamos unos tiempos de espera más largos, debemos utilizar el post-escalador seleccionando uno de los factores de división disponibles. Si seleccionamos un factor de 1:128, podemos llegar a periodos hasta de 2.3 segundos.

Registro: ESTADO

IRP	RP1	RP0	$\overline{TO}$	PD	Z	DC	C

***Bit* 7** ***Bit* 0**

Dirección: 03h
condición de *reset*: 000??XXX

IRP: Selector de página para direccionamiento indirecto
Este *bit* no se utiliza efectivamente en el PIC16F84, por lo que se puede utilizar como un *bit* de propósito general.

RP1,0: Selectores de página para direccionamiento directo.
Solamente RP0 se utiliza en el PIC16F84. RP1 se puede utilizar como un *bit* de propósito general.

TO: *Time Out* o *Bit* de finalización del temporizador.
Se coloca en 0 cuando el circuito de vigilancia *Watchdog* finaliza la temporización.

PD: *Power Down* o *Bit* de bajo consumo. Se coloca en 0 por la instrucción *SLEEP*.

Z: *Zero* o *Bit* de cero. Se coloca en 1 cuando el resultado de una operación lógica o aritmética es cero.

DC: *Digit Carry* o *Bit* de acarreo de dígito. En operaciones aritméticas se activa cuando hay acarreo entre el *bit* 3 y el 4.

C: *Carry* o *Bit* de acarrreo. En instrucciones aritméticas se activa cuando se presenta acarreo desde el *bit* más significativo del resultado.

Figura 16.5 *Registro de estado*

Modo Sleep

Cuando en un proceso en el cual interviene un microcontrolador este no opera durante algún período de tiempo, es posible llevarlo a un estado en el cual se desactivan algunas de sus funciones y consumirá menos energía. Esta característica es muy útil sobre todo en dispositivos autónomos o portátiles que requieran ahorrar al máximo el consumo de sus baterías.

Este modo de bajo consumo se llama **sleep**, que en inglés quiere decir "dormir"; de hecho se puede decir que el microcontrolador se pone a dormir. Cuando de nuevo se requiere que el microcontrolador funcione normalmente se debe despertar, para lo cual debe existir alguna de las condiciones que se citan más adelante.

Para entrar al modo **sleep** basta ejecutar la instrucción **SLEEP**; desde ese momento el oscilador se desactiva, se almacena el estado actual de todos los registros, los temporizadores se detienen (a excepción del TIMER1, si éste se encuentra funcionando con reloj externo), los puertos permanecen con el estado actual, el *bit* PD del registro de estado se pone en bajo y el *bit* TO se pone en alto, **figura 16.5**.

El **watchdog** sí continúa funcionando; de hecho cuando el dispositivo se encuentra en modo sleep puede ser despertado por el WDT. Para asegurar que el WDT está limpio, se debe ejecutar la instrucción **CLRWDT** antes de la instrucción **SLEEP**.

Figura 16.6 *Registro PCON*

Despertando del modo *sleep*

Dependiendo de cómo se despierta el PIC, el programa iniciará su operación en determinada posición de la memoria de programa; por ejemplo, si la salida del **sleep** se originó por una señal en el pin $\overline{MCLR}$ o por una señal del WDT, el PIC iniciará su operación en la dirección 0x000, que corresponde a la posición normal del *reset*. Si este es el caso, los *bits* $\overline{PD}$ y TO y el registro PCON pueden analizarse para identificar la causa de estar en la dirección de inicio, como lo veremos más adelante en el ejercicio de esta lección.

El registro **PCON** (*Power CONtrol*) contiene una serie de banderas que, junto con los *bits* TO y PD pueden utilizarse para determinar el origen del estado de *reset* del microcontrolador, **figura 16.6**.

El microcontrolador puede despertar del modo **sleep** por alguna de las siguientes razones:

1. *Reset* del dispositivo
2. Señal del WDT
3. Algún módulo periférico que pueda activar una bandera de interrupción mientras estamos en modo **sleep** como:
   - Pin externo INT
   - Cambio en un pin de entrada
   - Comparadores
   - A/D
   - *Timer1*
   - LCD
   - SSP
   - Captura

Un aspecto importante que se debe tener en cuenta dentro del modo **sleep**, es cuanto tiempo se demora un microcontrolador PIC para salir de él. Debido a que entre las funciones del PIC que se desactivan se encuentra el oscilador, cuando se pretende normalizar de nuevo el funcionamiento, se deben esperar 1.024 oscilaciones para asegurarnos que la señal es estable y a partir de ese momento empezar la ejecución normal.

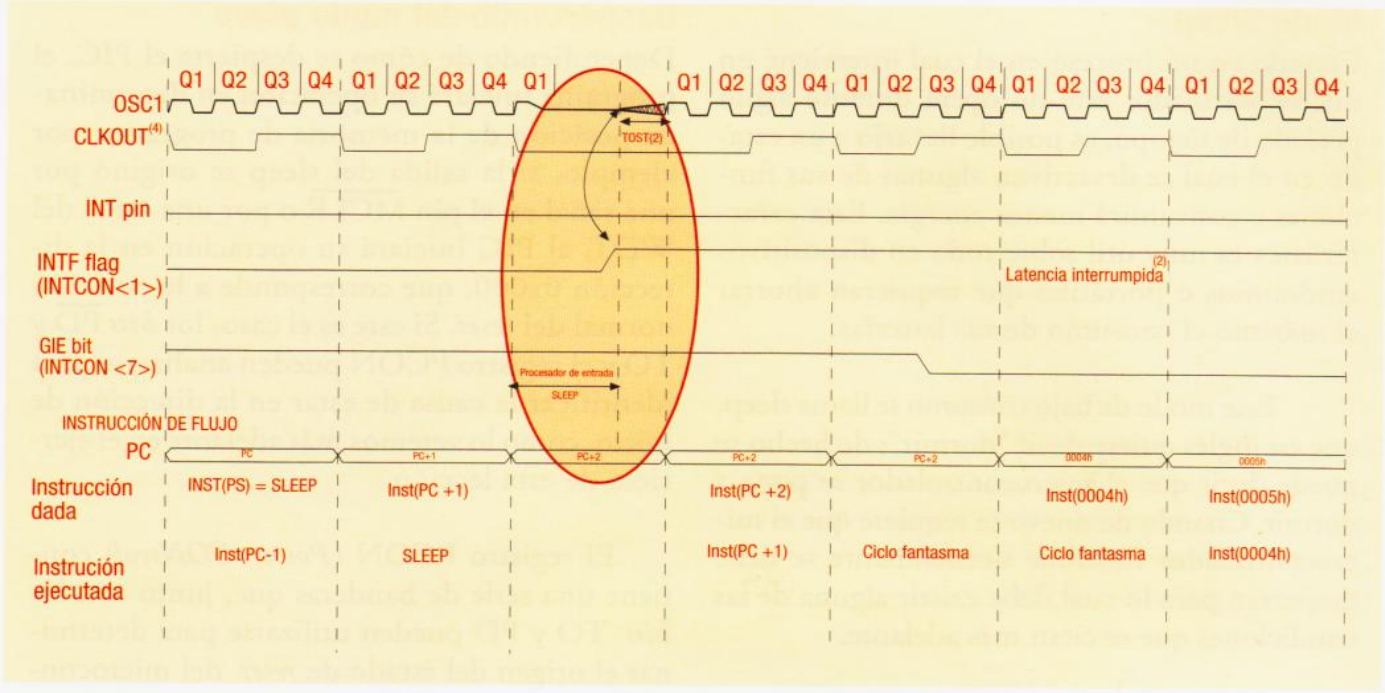


Figura 16.7 *Despertando mediante interrupciones*

También se pueden utilizar interrupciones para salir del estado de **sleep**. Si el *bit* GIE se encuentra en alto, el PIC saldrá de este estado hacia la dirección 04. Si esto no se debe a una interrupción, el PIC continuará el programa en el punto donde se detuvo antes de entrar al modo **sleep**.

El diagrama de tiempos que tiene lugar cuando se despierta por medio de una interrupción, se puede apreciar en la **figura 16.7**. Cuando se ejecuta la instrucción SLEEP siempre se busca la siguiente instrucción (PC+1); como siempre ocurre ese fenómeno, es conveniente colocar siempre después de la instrucción SLEEP una instrucción NOP.

Si observamos cuidadosamente la **figura 16.4** podemos apreciar como la ejecución de la instrucción **SLEEP** y la búsqueda de la instrucción en PC+1, se encuentran en el mismo espacio de tiempo. En PC+2 el procesador ya se encuentra en modo **sleep**. Si el *bit* GIE se encuentra habilitado y aparece la bandera que indica la aparición de una interrupción, el procesador espera un tiempo mientras se estabiliza la señal del reloj y de nuevo está en capacidad de ejecutar instrucciones.

Dependiendo de la causa que lo hizo despertar, ejecuta la instrucción en PC+2 o salta a la instrucción ubicada en la dirección 04.

SLEEP: Entra al modo dormido (*standby*)
Sintaxis: SLEEP
Descripción: El procesador se pone en modo SLEEP y se detiene su oscilador
Ejemplo: SLEEP

CLRWDT: Borra el contenido del temporizador **watchdog**
Sintaxis: CLRWDT
Descripción: Inicializa de nuevo el temporizador **watchdog**, al igual que el prescalador del WDT.
Ejemplo: CLRWDT
Antes de la instrucción Contador WDT =?
Después de la instrucción Contador WDT = 00
TO = 1, PD = 1

```
;*************************************
;EXPERIMENTANDO CON EL MODO SLEEP
;EN ESTE PROGRAMA PODEMOS OBSERVAR EL
;COMPORTAMIENTO DE LOS BITS PD  Y TO
;CUANDO EL MICROCONTROLADOR SE SALE DEL MODO SLEEP
;*************************************

     LIST P=16F84
     RADIX DEC
     INCLUDE  "C:\P16C84.INC"


     _CONFIG    WDT_OFF & _XT_OSC
                      ;PERRO GAURDIAN DESACTIVADO
;    _CONFIG    WDT_ON & _XT_OSC
                      ;PERRO GUARDIAN ACTIVADO

     ORG  0

     BSF  STATUS,RP0
     CLRF TRISA       ;POR EL PUERTO A
                      ;SALEN TO Y PD
     CLRF TRISB       ;LA DIRECCIÓN SALE
                      ;POR EL PUERTO B
     BCF  STATUS,RP0

LAZO COMF STATUS,W    ;MOSTRAR VALORES DE
                      ;TO Y PD
     MOVWF PORTA
     BSF  INTCON,INTE ;HABILITA
                      ;INTERRUPCIÓN POR RB0
     SLEEP            ;PONE EL PIC A
                      ;DORMIR
     NOP              ;PRECAUCIÓN PARA EL
                      ;SLEEP
     MOVLW 0X0FB      ;ESTA SALIDA INDICA
     MOVWF PORTB      ;QUE ESTAMOS EN MODO
                      ;SLEEP

     GOTO LAZO        ;NOS ENCASILLAMOS EN
                      ;UN CICLO

     END
```

Figura 16.8 *Ejercicio del manejo del WDT y el modo sleep*

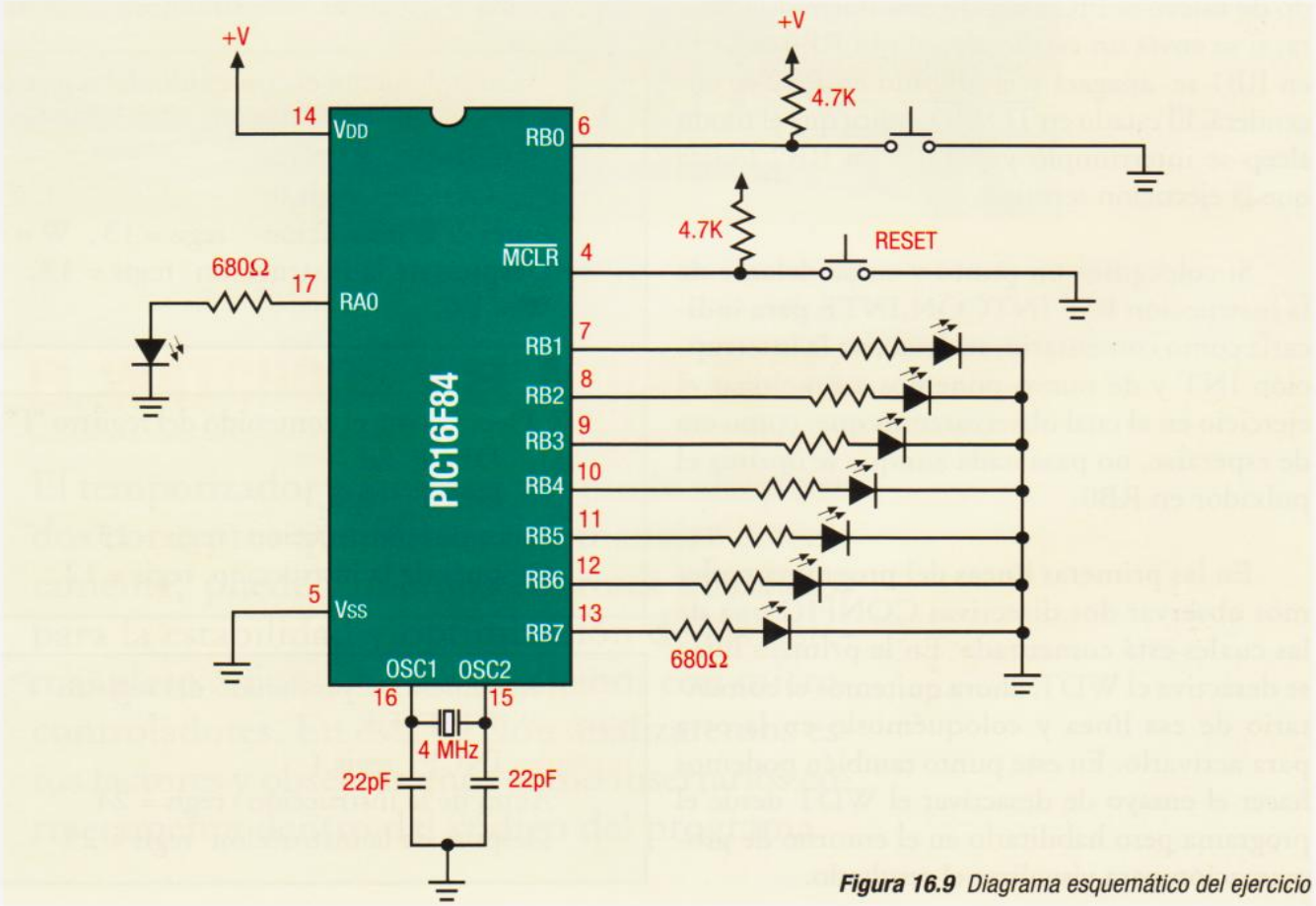


Figura 16.9 *Diagrama esquemático del ejercicio*

$\overline{TO}$	$\overline{PD}$	RB1	RB2	Descripción
1	1	1	0	*Power on reset*/MCLR durante la operación normal
1	0	1	0	Señal de _MCLR durante el modo **sleep**
1	0	0	1	Señal en RB0
0	0	0	1	Despertando por medio del WDT

Tabla 16.2 *Bits $\overline{PD}$ y $\overline{TO}$*

En el programa de la **figura 16.8** podemos analizar algunos detalles importantes de manejo del modo **sleep**; para ello observemos el diagrama esquemático de la **figura 16.9**. Inicialmente se despliega por el puerto el estado de los *bits* $\overline{TO}$ y $\overline{PD}$ que debe ser alto debido a que apenas se alimenta el sistema y el programa empieza en la dirección 0x0000. Se habilita la interrupción INT e inmediatamente se entra al modo **sleep**.

Si se entrega una señal de *reset,* la señal en PD desaparecerá debido a que se está inicializando de nuevo el PIC estando éste dormido. Ahora, si se envía un estado alto al pin RB0 el LED en RB1 se apagará y el ubicado en RB2 se encenderá. El estado en $\overline{TO}/\overline{PD}$ indica que el modo **sleep** se interrumpió y el LED en RB2 indica que la ejecución terminó.

Si colocamos un punto y coma delante de la instrucción BSF INTCON,INTE para indicarla como comentario, se desactiva la interrupción INT y de nuevo ponemos a funcionar el ejercicio en el cual observaremos que, como era de esperarse, no pasa nada aunque se oprima el pulsador en RB0.

En las primeras líneas del programa podemos observar dos directivas CONFIG una de las cuales está comentada. En la primera línea se desactiva el WDT, ahora quitemos el comentario de esa línea y coloquémoslo en la otra para activarlo. En este punto también podemos hacer el ensayo de desactivar el WDT desde el programa pero habilitarlo en el entorno de programación para visualizar el resultado.

Al alimentar el circuito podemos observar que los diodos LED que corresponden a $\overline{TO}$ y $\overline{PD}$ y RB1 se activarán. Si esperamos algunos segundos los tres LED se apagarán, pero el LED en RB2 permanecerá activado. Observemos en la **tabla 16.2** el resumen de los resultados obtenidos.

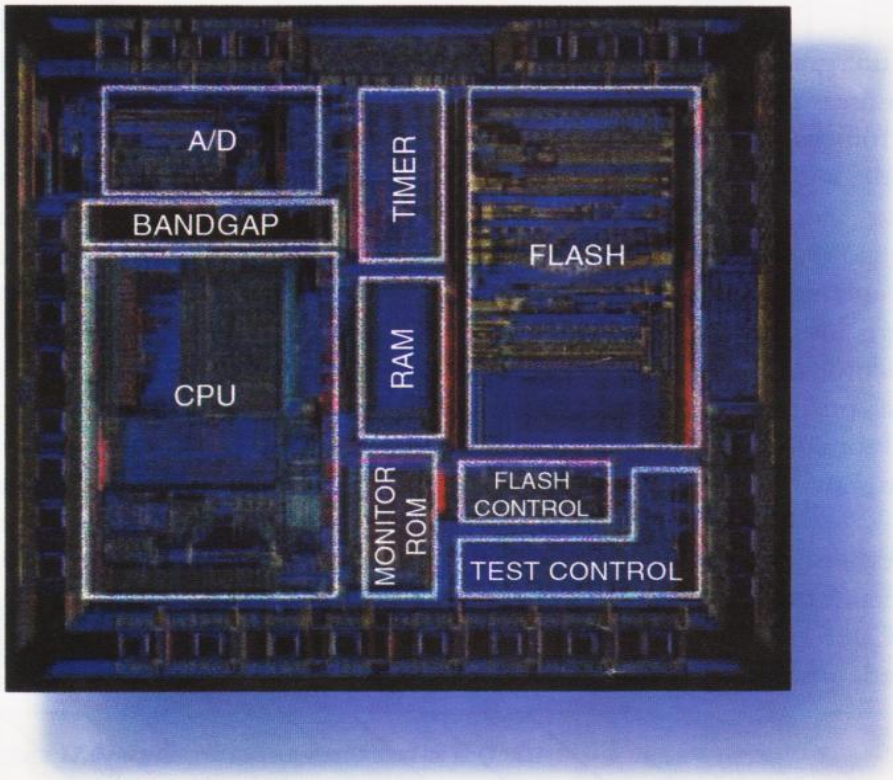
Este sencillo ejercicio puede ilustrarnos acerca de muchos detalles que se deben tener en cuenta para el manejos del **watchdog** y del modo **sleep**. Para experimentar, podemos montarlo en un *protoboard* y hacer toda clase de modificaciones al programa y visualizar los resultados.

COMF. Complementa el contenido del registro "f".
Sintaxis: COMF f,d
Ejemplo: COMF regis,0
Antes de la instrucción regis = 13 , W = ?
Después de la instrucción regis = 13, W = EC

DECF. Decrementa el contenido del registro "f".
Sintaxis: DECF f,d
Ejemplo: DECF regis,1
Antes de la instrucción regis = 13
Después de la instrucción regis = 12

INCF. Incrementa el contenido del registro "f".
Sintaxis: INCF f,d
Ejemplo: INCF regis,1
Antes de la instrucción regis = 24
Después de la instrucción regis = 25

LECCIÓN 17



ARQUITECTURA DE LOS MICROCONTROLADORES MOTOROLA

En esta lección estudiaremos la estructura interna o arquitectura, y el flujo de información entre la unidad central y los periféricos de los microcontroladores de la familia 68HC08, los cuales son muy representativos dentro de la amplia gama de estos circuitos integrados, fabricados por MOTOROLA.

Desde la aparición de los primeros microcontroladores fabricados por la empresa estadounidense MOTOROLA de la familia 68XX, su popularidad ha ido creciendo a pasos agigantados en virtud de sus magníficas características, tales como inmunidad al ruido eléctrico, facilidad de programación, velocidad, compatibilidad, flexibilidad, y una amplia gama de características y referencias disponibles.

Ellos han contado con una gran acogida en la industria especialmente en áreas como: el desarrollo de dispositivos electrónicos y sistemas de comunicación telefónica analógica o digital, las redes de transmisión de datos, vídeo y sonido, los sistemas de rastreo y localización satelitales (*global position system* GPS), los sistemas inteligentes para el control del tráfico terrestre, los dispositivos digitales para el almacenamiento, reproducción y reconocimiento de voz, y la construcción de dispositivos para el control automático de algunas funciones dentro de los automóviles modernos.

Actualmente los microcontroladores de Motorola cuentan con cinco familias principales: la **68HC05**, la **68HC08**, la **68HC11**, la **68HC12** y la **68HC16**, cada una con una unidad central de procesamiento (CPU) particular y un amplio y completo conjunto de instrucciones. Adicionalmente, cuentan con un excelente nivel de compatibilidad de código entre todas sus familias, lo cual los hace de fácil uso y flexibles, permitiendo el intercambio de programas entre microcontroladores de diferentes familias.

Entre las diferencias más significativas de estas cinco familias se encuentran: las nuevas instrucciones, la longitud de palabra (8 ó 16 *bits*), la disponibilidad de temporizadores (1 a 8 *timers*), los tipos de puertos de comunicación (SCI, SPI, I2C, MBUS, USB, SIOP y CAN), el espacio y tipo de memoria (EEPROM, FLASH y RAM), el número de canales A/D (4 a 10 canales), y el número disponible de puertos digitales de entrada y salida (entre 15 y 91).

En este curso se estudiarán tres de las familias más populares y representativas de los microcontroladores de Motorola, la 68HC08, la 68HC11 y la 68HC12, que tienen una gran acogida por parte de los fabricantes y estudiantes de esta tecnología en muchas industrias, centros educativos y universidades del mundo; debido a su disponibilidad, variedad de ca-

Dispositivo	Bits	ROM (Bytes)	RAM (Bytes)	FLASH/ (Bytes) OTP	EEPROM (Bytes)	Temporizadores	I/O	Puerto serial	A/D (8 bits)	PWM	COP (Watchdog)	Voltaje de operación (V)	Máxima frecuencia del bus (M)	Encapsulado
MC68HC908AS60	8	-	2K	60K FLASH	1K	2 (16 Bits)	45/50	SCI/SPI	15 canales	TIM	Y	5,0	8,0	52 PLCC/64 QFP
XC68HC08AZ0	8	-	1K	-	512	2 (16 Bits)	48	SCI/SPI	8 canales	TIM	Y	5,0	8,0	100 LQFP
XC68HC08AZ32	8	32K	1K	-	512	2 (16 Bits)	48	SCI/SPI	8 canales	TIM	Y	5,0	8,0	64 QFP
XMC68HC908AZ60	8	-	2K	60K FLASH	1K	2 (16 Bits)	48	SCI/SPI	15 canales	TIM	Y	5,0	8,0	64 QFP
XC68HC908GP20	8	-	512	20K FLASH	-	2 (16 Bits)	33	SCI/SPI	8 canales	TIM	Y	3,0 - 5,0	8,0	40 DIP
MC68HC908GP32	8	-	512	32K FLASH	-	2 (16 Bits)	33	SCI/SPI	8 canales	TIM	Y	3,0 - 5,0	8,0	40 DIP
MC68HC90	8	-	128	4K FLASH	-	2 (16 Bits)	23	-	12 canales	TIM	Y	3,0 - 5,0	8,0	28DIP
MC68HC908JK1	8	-	128	1.5K FLASH	-	2 (16 Bits)	15	-	10 canales	TIM	Y	3,0 - 5,0	8,0	20 DIP
MC68HC908JK3	8	-	128	4K FLASH	-	2 (16 Bits)	15	-	10 canales	TIM	Y	3,0 - 5,0	8,0	20DIP
XC68HC08LN56	8	56K	128	-	-	2 (16 Bits)	42	SCI/SPI	4 canales	TIM	Y	3,3 - 5,0	8,0	144 LQFP
XC68HC708MP16	8	-	512	16K OTP	-	2 (16 Bits)	44	SCI/SPI	10 canales	TIM	Y	5,0	8,0	64 QFP
XC68HC908MR24	8	-	512	24K FLASH	-	2 (16 Bits)	44	SCI/SPI	10 canales	TIM	Y	5,0	8,0	64 QFP

Tabla17.1 *Cuadro de características principales de los microcontroladores Motorola de la familia 68HC08*

racterísticas, reducido costo y facilidad de programación y depuración por medio de sus herramientas de desarrollo.

En la **tabla 17.1** se muestran de manera resumida las principales características de algunos de los microcontroladores más populares de la familia 68HC08 de Motorola.

La familia 68HC08

Los microcontroladores MC68HC908JXX son miembros de la familia 68HC08 de rango medio, bajo costo y alto desempeño. Todos los microcontroladores de esta familia poseen una palabra de un *byte* (8 *bits*), y usan la misma unidad central de procesamiento conocida como CPU08. Las principales características con las que cuentan, y que son típicas para la mayoría de los microcontroladores de Motorola son:

- CPU08 con arquitectura Von Neumann y bus interno de 8MHz.
- Amplio y completo conjunto de instrucciones (poseen más de 90 instrucciones con funciones simples y especializadas).
- Código de programa y tecnología compatible con familias de rango inferior, tales como las M6805, HC05 y M146805.
- Diseño CMOS con bajo consumo de potencia y con un rango de alimentación entre 3 y 5,5VCC, ideal para la fabricación de dispositivos portátiles.
- Dos tipos de osciladores: con red RC o con cristal de cuarzo externos.
- Memoria FLASH para el usuario de 1.536 ó 4.096 *bytes*, con estrategia de seguridad que dificulta la lectura o copia no autorizada de los datos contenidos en ella.
- Memoria RAM con capacidad entre 128 y 512 *bytes*.
- Dos módulos de temporización de 16 *bits* cada uno (TIMA y TIMB).
- Doce canales, cada uno con convertidor análogo/digital de 8 *bits*.
- Quince líneas de entradas y salidas digitales de propósito general, dentro de las que se cuentan las interrupciones para el manejo de teclado con resistencias internas de *pull-up*, los *drivers* o amplificadores de corriente para diodos LED, las entradas o salidas de drenador abierto (*open-drain*) de 25mA cada una, las líneas para trabajar en el modo de captura (ICAP/OCAP), la salida del generador de PWM.
- Cuatro sistemas de protección para el código del programa, que son: COP (sistema de vigilancia *Watchdog*), LVI (inhibidor de bajo voltaje), detector de código ilegal y detector de dirección ilegal.
- Sistema maestro de restablecimiento (*RESET*) activo en nivel bajo, con resistencia interna de sujeción a Vdd, e interrupción de restablecimiento a la conexión de la alimentación POR (*power on reset*).

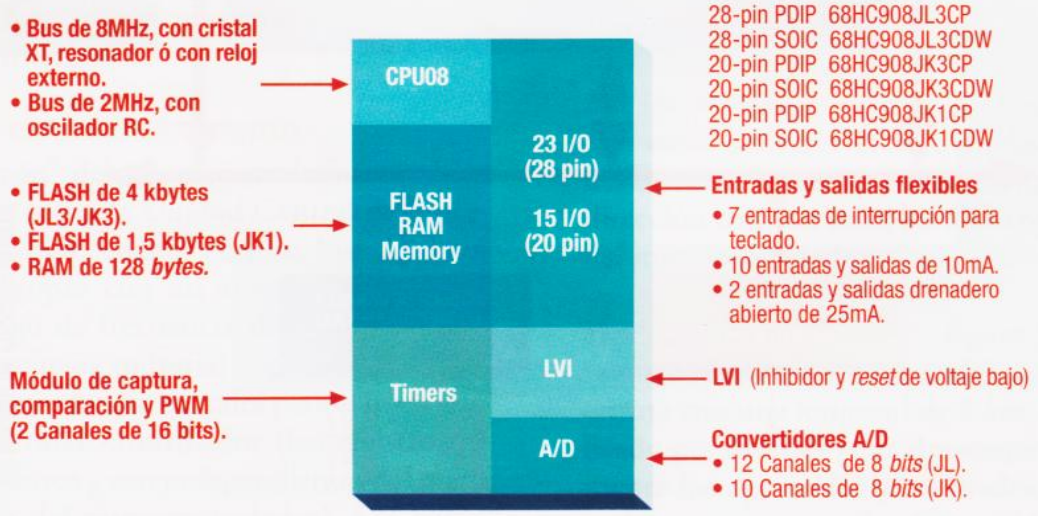


Figura 17.1 *Sumario de las características principales dentro de los microcontroladores de la familia 68HC08*

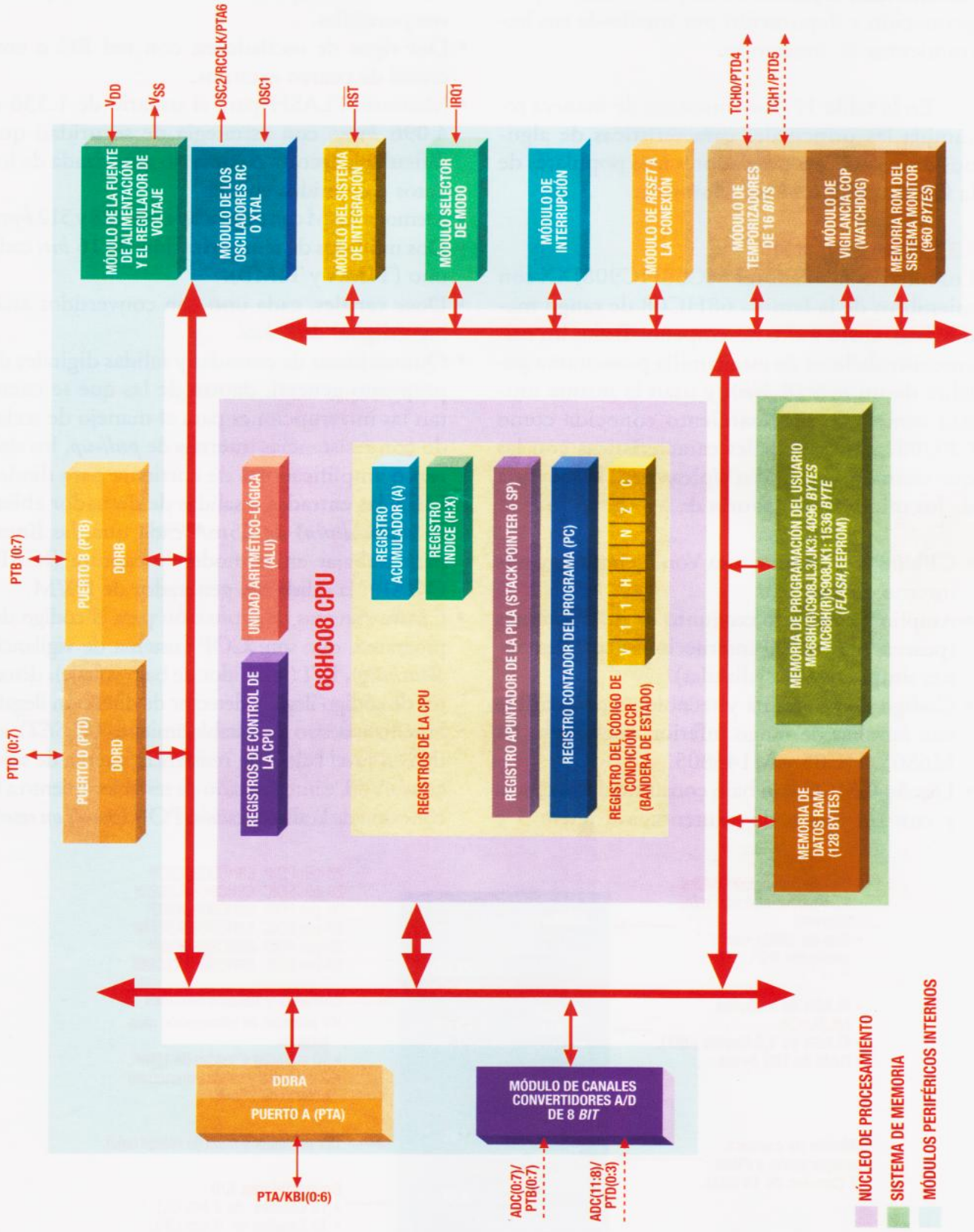


Figura 17.2 *Diagrama en bloques que muestra la arquitectura de los microcontroladores de la familia 68HC08*

- Una entrada de interrupción maestra no sincrónica IRQ activa en nivel bajo, con resistencia de sujeción y tecnología *schmitt-trigger*, programables.

La **figura 17.1** muestra de manera simplificada las principales características de la familia de microcontroladores 68HC08, y los principales subsistemas en los que se divide.

Arquitectura de los microcontroladores Motorola

En el diagrama en bloques que se muestra en la **figura 17.2** vemos la estructura, disposición interna e interconexión que hay entre las diferentes partes que conforman un microcontrolador Motorola típico de rango medio, tal como los de la familia 68HC08.

En la figura se puede apreciar con claridad la manera como están interconectados los principales bloques del microcontrolador; allí puede verse que entre todas sus partes hay un flujo de información bidireccional que se hace por medio de un sistema de bus único, tal como en la arquitectura Von Neumann, y que está controlado y sincronizado desde la CPU.

Todo microcontrolador puede dividirse en tres grandes subsistemas, que en el caso de los Motorola se pueden apreciar claramente en el diagrama de bloques y son: el núcleo de procesamiento, el sistema de memoria y los módulos periféricos internos.

El núcleo de procesamiento

Es el "corazón" del microcontrolador, también se le conoce como la Unidad Central de Procesamiento (CPU); cuenta con un bus de datos interno y propio con un ancho de banda de 8MHz (rango de frecuencia dentro del cual el bus tiene ganancia unitaria) y contiene los módulos y registros fundamentales para el buen desempeño del microcontrolador (los registros de la CPU son cinco y son independientes del mapa de memoria del microcontrolador). Las partes que conforman la CPU son:

1. La unidad aritmético-lógica ALU, que está representada por un conjunto de registros de propósito particular e instrucciones especializadas disponibles para el usuario; está constituida por un módulo sumador completo (con acarreo intermedio y final) de 8 *bits*, un multiplicador de 8 *bits* por 8 *bits*, y un módulo de división binario de 16 *bits* en 8 *bits*. La tarea de la ALU, como su nombre lo indica, consiste en procesar todas las operaciones aritméticas o lógicas que debe hacer la CPU, tales como las sumas, divisiones, multiplicaciones, rotaciones y desplazamientos, entre otras.

   Sin embargo, cumple también un papel fundamental para poder acceder a los datos disponibles en memoria directa, o indirectamente a través del registro contador del programa (PC), cuando se ejecutan instrucciones de salto, llamados y retornos de otras funciones o de subrutinas e interrupciones.

   De su velocidad de ejecución y buenas características técnicas se desprende la disponibilidad que ofrece el microcontrolador para efectuar cálculos con grandes volúmenes de datos, de una manera ágil y segura, tal como se requiere en las aplicaciones de manejo y reconocimiento de voz y vídeo.

2. El registro de control de la CPU, que está conformado por un grupo de *bits* que generan las señales de sincronización y control necesarias para que el flujo de datos a través del bus se efectúe sin contratiempos y en la dirección correcta entre los diferentes bloques del microcontrolador.

3. El registro acumulador A, **figura 17.3**, es un registro de propósito general de lectura y escritura con una longitud de 8 *bits*, que es utilizado por la CPU para almacenar temporalmente los operandos y resultados obtenidos en las operaciones aritméticas y lógicas realizadas por la ALU.

Figura 17.3 *Registro acumulador de la CPU08*

4. **El registro índice H:X**, **figura** 17.4, es un registro de 16 *bits* que permite direccionar hasta 64 *kbytes* de espacio en memoria en forma indexada. El *byte* más alto del registro índice es H, mientras que el *byte* más bajo es X; de esta forma se construye el registro concatenado de 2 *bytes* H:X que es el índice para el área de memoria del usuario, y con el cual se puede apuntar a cualquiera de las 64.000 posiciones.

   Este registro también se puede usar como uno de propósito general para el almacenamiento temporal de datos; sin embargo se debe recordar que la CPU usa su contenido para determinar la dirección efectiva del operando, y además, tener presente que la parte alta del registro se borra tras la acción de restablecimiento o de *reset* en el microcontrolador (H=0).

Figura 17.4 *Registro índice de la CPU08*

5. **El registro apuntador de la pila SP (*stack pointer*)**, **figura** 17.5, es un registro de 16 *bits* que contiene en cualquier instante la dirección de la próxima posición de 8 *bits* dentro de la pila del microcontrolador. En los microcontroladores Motorola el registro SP se puede usar como el índice para acceder a los datos temporales dentro de la pila, en dos modos de direccionamiento.

   La dirección contenida en el SP disminuye en uno con cada dato que entra a la pila, y aumenta en uno cada que un dato es extraído de la misma; de esta manera el SP siempre apunta a la próxima posición disponible en la pila.

   Es importante recordar que luego de la ejecución del ciclo de *reset* maestro, en el SP se graba la dirección hexadecimal 00FF, para proveer compatibilidad con la pila de la familia MC6805; mientras que con la instrucción RSP (*reset stack pointer*), se restablece el registro SP colocando su *byte* más bajo en FF y no se afecta el *byte* más significativo.

Figura 17.5 *Registro apuntador de la pila de la CPU08*

PARA SABER MÁS

Movilidad de la pila (*stack*)

En la arquitectura de los microcontroladores Motorola de las familias 68HC08 y superiores, la ubicación de la pila no es fija, es arbitraria, y puede relocalizarse en cualquier espacio dentro de la memoria RAM. Para esto, solo basta grabar el registro apuntador de la pila SP con una dirección de memoria diferente a la de la página 0 (0000 a 00FF), que es el lugar por defecto en el que reside la pila.

Aunque el desplazamiento y la ubicación de la pila sea libre para el programador, y a pesar de que el registro SP es de 16 *bits*; para un adecuado funcionamiento nunca se debe olvidar que el apuntador de la pila debe permanecer dentro del área de la memoria RAM.

6. El registro contador del programa PC (*program counter*), **figura 17.6**, es también un registro de 16 *bits* que contiene en todo momento la dirección de la próxima instrucción o el operando que se debe buscar, a través de un ciclo de búsqueda de la CPU. La CPU siempre aumenta el valor del PC en forma secuencial hacia la siguiente posición de memoria, cada vez que una instrucción o un operando sea buscado; excepto cuando se ejecutan operaciones de salto, llamado, retorno e interrupciones, que graban direcciones particulares en el PC.

   Durante el ciclo de restablecimiento, el registro PC es grabado con el contenido del vector del *reset* ubicado en las posiciones de memoria FFFE y FFFF; de esta forma queda cargado con un número de 16 *bits*, que representa la dirección de la primera instrucción del programa que debe ser ejecutada al terminar este ciclo.

Figura 17.6 *Registro contador del programa de la CPU08*

7. El registro del código de condición CCR (*condition code register*), **figura 17.7**, es también llamado por otros fabricantes de microcontroladores como registro de estado o registro de banderas de estado; su longitud es de 8 *bits* y contiene el *bit* habilitador de interrupciones y 5 *bits* más que indican el resultado de la instrucción que está siendo decodificada. Tras la ejecución del ciclo de *reset* el contenido del registro CCR solo se afecta en el *bit* 3, que es el habilitador de interrupciones del microcontrolador (I); éste se coloca en 1 lógico y las inhabilita.

El sistema de memoria

El sistema de memoria en un microcontrolador es de gran importancia ya que está destinado a contener el código del programa con todas sus variables y constantes. La flexibilidad, confiabilidad, vida útil y facilidad de manejo, son factores importantes para el usuario de esta tecnología y que están ligados con el tipo de memoria disponible para el usuario en el microcontrolador.

V - **Bit indicador de sobreflujo (*overflow flag*).** Es colocado en 1 lógico cuando se presenta un sobreflujo o desbordamiento en el rango de una operación aritmética, en complemento a dos.

H - ***Bit* indicador de acarreo intermedio (*half-carry flag*).** Es colocado en 1 lógico cuando ocurre un acarreo entre los bits 3 y 4 del registro acumulador, durante una operación aritmética.

I - ***Bit* habilitador de interrupciones (*interrupt mask*).** Cuando este bit es colocado en 1 lógico, todas las interrupciones enmascarables de la CPU se inhabilitan; mientras que cuando este *bit* se borra (en 0 lógico) se habilitan. Este *bit* es automáticamente colocado en 1 luego de la acción de *reset* y la atención a una señal de interrupción.

N - ***Bit* indicador negativo (*negative flag*).** Este *bit* es colocado en 1 lógico cuando el resultado de una operación aritmética o lógica, o del procesamiento de datos, es negativo.

Z - ***Bit* indicador de cero (*zero flag*).** Cuando el resultado de una operación lógica, aritmética o entre datos es 00, este *bit* es colocado en 1 lógico; de lo contrario se mantiene en 0 lógico.

C - ***Bit* indicador de acarreo (*carry/borrow flag*).** La CPU coloca este *bit* en 1 lógico cuando se presenta un acarreo hacia afuera o hacia adentro del séptimo *bit* del acumulador, durante la ejecución de las operaciones de adición y sustracción, y algunas operaciones lógicas.

Figura 17.7 *Registro del código de condición de la CPU08*

En el caso de los microcontroladores Motorola, el usario cuenta opcionalmente con tres tipos de memorias que pueden ser borradas o grabadas en cualquier instante (FLASH, EEPROM y EPROM), y otros dos tipos que solamente se pueden grabar una vez (OTP y ROM).

La organización de la memoria, sus características y sus modos de direccionamiento, por ser de gran importancia, se tratarán con más detalle en una futura lección, sin embargo ahora se puede decir que, al igual que en muchos otros microcontroladores, ésta se encuentra dividida en dos grandes bloques, que son: la memoria de programa y la memoria de datos, y su conexión con la CPU se hace a través del mismo sistema de bus.

Los módulos periféricos internos

Los módulos periféricos internos desempeñan funciones básicas para el buen funcionamiento de la CPU, como en el caso de los módulos de reloj, las interrupciones, el *reset*, la regulación y protección a la alimentación; ellos facilitan la operación de todo el sistema y la conexión con otros dispositivos o sistemas del mundo exterior. Además, desarrollan tareas específicas dentro de la arquitectura de un microcontrolador y los orientan hacia aplicaciones particulares dentro del campo de la industria y la experimentación con esta tecnología.

Los microcontroladores Motorola, dentro de la familia 68HC08, cuentan con módulos periféricos versátiles de mucho uso y fácil manejo, que agilizan las tareas de control y procesamiento de datos que se pueden desarrollar. Los módulos periféricos o interfaces que ofrece una familia de rango medio como la 68HC08, son: *drivers* digitales para diodos LED, *drivers* de drenador abierto para cargas de alto consumo, interrupciones para teclado, contadores y comparadores de captura (ICAP/OCAP), temporizadores programables (TIMX), puertos de comunicación serial (SCI, SPI y CAN), canales de conversión A/D, y salidas de control PWM.

Las características y funciones de los módulos periféricos de mayor importancia para el microcontrolador y el usuario, serán tratadas ampliamente en las próximas lecciones.

Asignación de pines en los microcontroladores 68HC08

Los microcontroladores Motorola de rango medio de la familia 68HC08 están disponibles en encapsulados tipo DIP de 20 y 28 pines. En la **figura 17.8** se muestra la distribución y asignación de los pines de los microcontroladores JK1, JK3 y JL3.

Los microcontroladores JL3 poseen, a diferencia de los JK1 y JK3, un mayor grupo de líneas disponibles(puertos PTA0 a PTA5 y puertos PTD0 y PTD1).

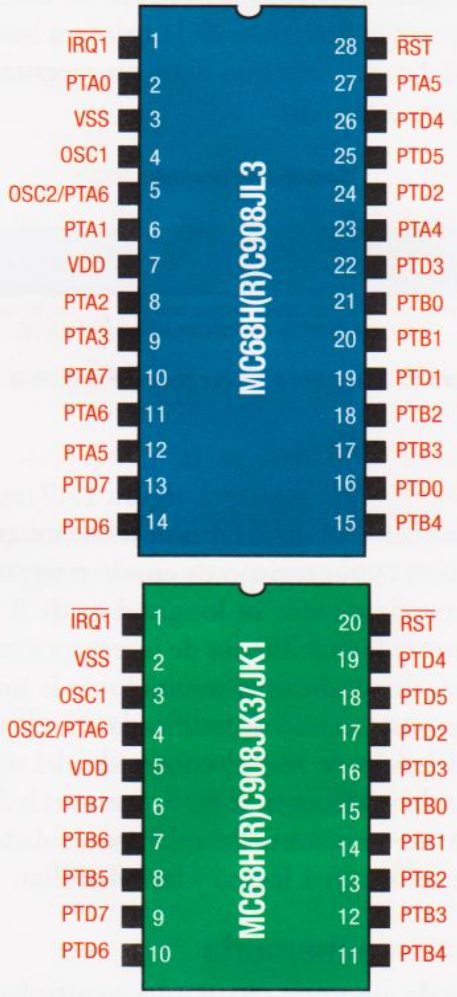
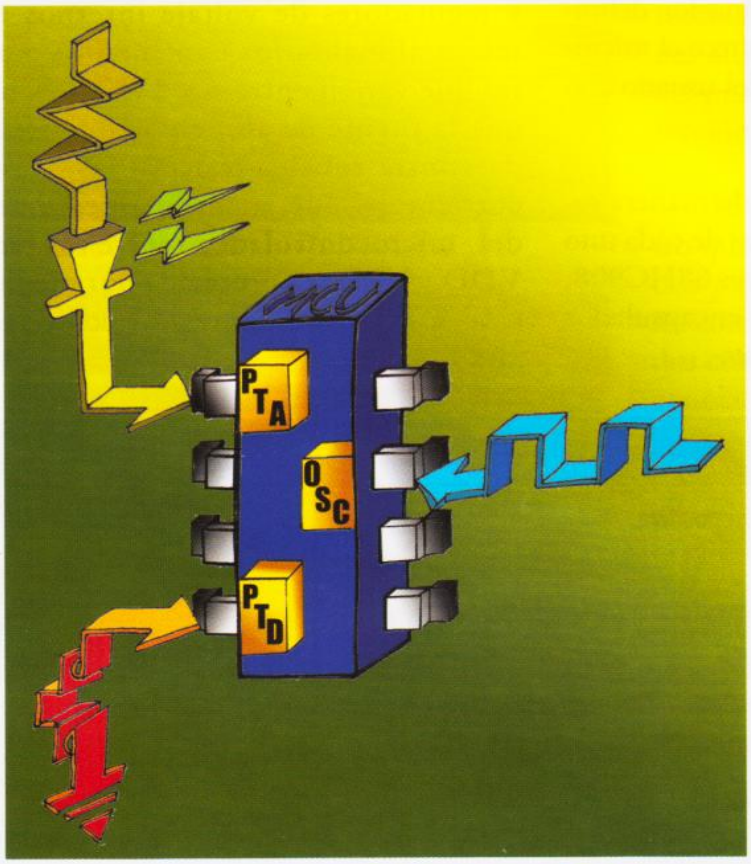


Figura 17.8 Asignación de pines de los microcontroladores 68HC08

LECCIÓN 18



LOS PUERTOS DE LOS MICROCONTROLADRES HC08 DE MOTOROLA

Dentro de los módulos con los que cuentan los microcontroladores de esta familia, los puertos de entrada y salida (I/O) son de gran importancia para la interconexión y el manejo de las cargas y dispositivos externos a ellos. En la actualidad los microcontroladores de rango y velocidad media, evolucionan en torno a la especialización de las tareas que son capaces de desarrollar, y esto depende de las funciones y características que los diseñadores hayan incluido en los puertos.

Descripción de los pines y sus funciones

Para continuar conociendo la estructura interna y los módulos que rigen el funcionamiento de los microcontroladores Motorola de la familia **HC08**, se debe conocer la distribución y asignación definida para cada uno de los *pines* que ofrece el microcontrolador para la interconexión del usuario con el mundo externo.

En la **tabla 18.1** se describen de manera resumida las principales características de cada uno de los *pines* de los microcontroladores 68HC908, en sus versiones JK1, JK3 y JL3, y encapsulados en estructuras DIP de 20 y 28 *pines*.

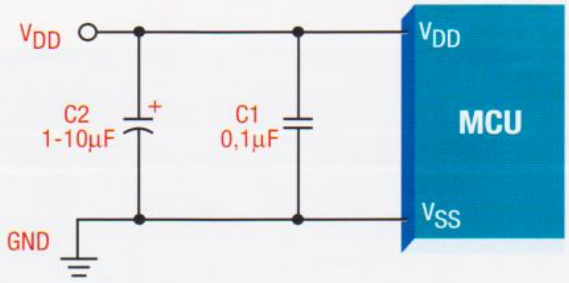
La alimentación del microcontrolador es uno de los factores más importantes para que el circuito integrado y todos sus módulos internos trabajen adecuadamente, y aunque la tecnología usada por Motorola, posee filtros y reguladores de voltaje internos que permiten alimentarlos con fuentes monopolares que varíen entre 3 y 5VCC, es necesario que la fuente de alimentación esté protegida contra señales transitorias de voltaje y corriente, y que la entrada de alimentación del microcontrolador (entre los terminales VDD y VSS) sea protegida por medio de uno o varios condensadores no polarizados para alta frecuencia.

Nombre del pin	Pin	Descripción del pin	Dirección	Nivel de voltaje
VDD		Entrada de alimentación positiva.	Entrada	3V ó 5V
VSS		Salida de cero voltios, usado como referencia para la fuente de alimentación (sistema de tierra ó GND).	Salida	0V
$\overline{RST}$		Entrada externa de inicialización de nivel activo bajo ó en cero lógico. Tiene resistencia de *pull-up* y entrada conformadora de señal a través de compuertas *schmitt trigger*.	Entrada	VDD
$\overline{IRQ1}$		Entrada externa activa en nivel bajo, definida para la petición del servicio de interrupción. Posee una resistencia de *pull-up* y una compuerta de entrada *schmitt trigger* opcionales y programables por medio del código del programa (*software*). Este *pin* cumple también una función alterna como entrada del selector de modo del microcontrolador.	Entrada	VDD a $VDD+V_{HI}$
OSC1		Entrada del módulo oscilador con base en cristal de cuarzo XTAL, resonador u oscilador pasivo RC.	Entrada	Análogo
OSC2		Este *pin* puede operar como una entrada o como una salida, dependiendo del tipo de oscilador configurado. Cuando se usa el cristal de cuarzo o el resonador, el *pin* se comporta como salida y la señal que se obtiene en él, es la señal invertida que entra al módulo oscilador OSC1. Cuando se elige la opción de la red pasiva RC en el oscilador el *pin* puede ser una entrada o una salida, y por defecto, se comportará como la salida del reloj RCCLK. Adicionalmente esta entrada se comparte con las funciones del puerto A.6 y de la sexta interrupción para el teclado KBI6, y tiene una resistencia de *pull-up* programable.	Entrada/ Salida	Análogo/ VDD

Nombre del pin	Pin	Descripción del pin	Dirección	Nivel de voltaje
PTA[0:6]	JK:4 JL:2,5, 6,8,9, 23,27	Las líneas de este conjunto de 7 *bits,* que conforman el puerto A, pueden ser de entradas o salidas de propósito general y también pueden usarse como siete entradas de interrupción, y para el manejo de teclados KBI[0:6]. En este puerto todos los *pines* pueden ser programados individualmente con resistencia interna de *pull-up.*	Entrada/ Salida	VDD
PTB[0:7]	JK:6 a 8,11 a 15 JL:10 a 12,15, 17,18, 20,21	Las líneas de este conjunto de 8 *bits* que conforman el puerto B, se comportan como entradas o salidas de propósito general, y también comparten su función con ocho canales de los convertidores A/D con que cuenta el microcontrolador, ADC[0:7].	Entrada/ Salida	VDD/ Análogo
PTD[0:7]	JK:9, 10,16 a 19 JL:13, 14,16, 19,22, 24 a 26	Este conjunto de 8 *bits* bidireccionales de propósito general, constituye el puerto D. Su función está compartida con cuatro canales A/D (ADC8 a ADC11) del microcontrolador, a través de los puertos PTD[0:3]; y con los dos canales bidireccionales del módulo de la interfaz de temporización TIM, TCH0 y TCH1. Adicionalmente, los *pines* de los puertos D.6 y D.7 podrán ser configurados como entradas y salidas digitales, con capacidad de 25mA, interfaz tipo drenador abierto, y resistencia *pull-up* interna.	Entrada/ Salida	VDD/ Análogo

Nota:
En los microcontroladores con encapsulado tipo DIP de 20 *pines* JK, los terminales PTA0, PTA1, PTA2, PTA3, PTA4, PTA5, PTD0 y PTD1 no se encuentran disponibles.

Tabla18.1 *Descripción general de las características, funciones y niveles de voltaje de los* pines *en los microcontroladores Motorola de la familia 68HC08.*



Nota: (Los valores de los condensadores son típicos)
- **C1** debe ser un condensador no polarizado y para alta frecuencia (cerámico o de poliester).
- **C2** es un condensador opcional electrolítico que se requiere cuando los puertos del microcontrolador entregan toda su carga.

Figura 18.1 *Filtrado externo de la entrada de alimentación del microcontrolador*

Durante la energización inicial, debido a la excitación de grandes cargas a través de sus puertos, al *reset*, o a la entrada de señales transitorias de alta frecuencia (ruido eléctrico) provenientes de fuentes externas, el microcontrolador demanda una gran cantidad de corriente de la fuente en forma transitoria, que pueden dañarlo u ocasionar su mal funcionamiento.

Colocando el conjunto de condensadores C1 y C2 tal como se muestra en la **figura 18.1**,y lo más cerca posible de los *pines* de alimentación, se crea un filtro pasa bajos simple que puede pro-

teger efectivamente al microcontrolador, suprimiendo las señales de alta frecuencia antes de que puedan afectarlo.

Los puertos de entrada y salida (I/O)

Los puertos de entrada y salida I/O, permiten que el microcontrolador se conecte de manera segura, y maneje con niveles de corriente y voltaje adecuados todos los dispositivos y sistemas periféricos externos.

Los microcontroladores Motorola de 8 *bits* que pertenecen a la familia HC08, poseen 23 puertos bidireccionales de entrada y salida muy versátiles. Todos los puertos pueden ser programados en varios modos y efectuar diferentes funciones, tal como se estudia a continuación.

El puerto A

A este puerto se puede acceder a través del registro de datos A (PTA), y está compuesto por 7 *bits* de lectura y escritura que poseen funciones especiales de entrada y salida digitales. También puede ser configurado como 7 entradas externas de interrupción para el manejo de teclados (**KBI[6:0]**). Todos los terminales del puerto poseen una resistencia interna de *pull-up* que puede ser conectada o desconectada por medio del código del programa, cuando se configuran como entrada.

En este puerto los terminales entre **PTA0** y **PTA5** del registro de datos poseen fuentes limitadas de corriente orientadas al manejo de diodos LED directamente, o sea que no es necesaria la resistencia limitadora en serie con él.

Por medio del registro de dirección de datos del puerto A (**DDRA**), se programa la condición de entrada o salida sobre cada terminal o *pin* del puerto; con un uno lógico en cualquiera de sus *bits* se programa el puerto correspondiente como salida, mientras que con un cero lógico, se deshabilita el *buffer* de salida y se programa como entrada.

PARA SABER MÁS

Destino de los puertos no usados

Cuando se usa un microcontrolador en el desarrollo de un sistema, dispositivo o experimento, es posible que la tarea no requiera de todos sus terminales disponibles como puertos; en estos casos, éstos deben ser terminados, es decir, conectados a un nivel lógico apropiado tal como 0V (VSS) ó 5VCC (VDD).

Aunque los puertos de los microcontroladores Motorola pueden dejarse en circuito abierto cuando no se usan (puertos no terminados), y no se requiere cerrar el circuito para su buen desempeño, es recomendable, por su tecnología basada en interruptores electrónicos tipo FET (*field effect transistor*), que sean terminados para evitar el consumo excesivo de corriente y la posibilidad de que se dañen permanentemente por efecto de las cargas electrostáticas.

En cualquier momento, la acción del *reset* afectará al registro **DDRA**, borrando todos sus *bits* de control, y programando todo el puerto A como entrada. En la **figura 18.2** se resumen las direcciones, las características y las funciones alternas propias del registro de datos del puerto A (**PTA**) y del registro de dirección de datos A (**DDRA**).

Dirección del registro PTA: $0000

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura:	0	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0
Escritura:								
Reset:	No es afectado							
Funciones adicionales:			LED (Sink)	LED (Sink)	LED (Sink)	LED (Sink)	LED (Sink)	LED (Sink)
		30k pull-up	30k pull-up	30k pull-up	30k pull-up	30k pull-up	30k pull-up	30k pull-up
		Interrupción de teclado	Interrupción de teclado	Interrupción de teclado	Interrupción de teclado	Interrupción de teclado	Interrupción de teclado	Interrupción de teclado

Dirección del registro DDRA: $0004

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura:	0	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
Escritura:								
Reset:	0	0	0	0	0	0	0	0

Figura 18.2 *Definición de los registros PTA y DDRA*

Dirección del registro PTAPUE: $000D

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: / Escritura:	PTA6EN	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
Reset:	0	0	0	0	0	0	0	0

PTA6EN: ***bit* habilitador del pin PTA6 (válido solo con oscilador RC)**
***Bit* en 1:** configura el segundo pin del oscilador (OSC2) como un puerto de entrada y salida de propósito general.
***Bit* en 0:** configuración por defecto del pin OSC2 como la salida del reloj del oscilador RC (RCCLK).
PTAPUE[6:0]: ***bits* habilitadores de las resistencias *pull-up* del PTA**
***Bit* en 1:** configura la resistencia *pull-up* de 30k
***Bit* en 0:** desconecta la resistencia *pull-up*

Figura 18.3 *Definición del registro PTAPUE*

El puerto A cuenta además con el registro habilitador de las resistencias *pull-up* internas, llamado **PTAPUE**, con el cual se pueden conectar o desconectar dichas resistencias. En cada *bit* del puerto puede configurarse una resistencia de *pull-up* independientemente, siempre y cuando éstos estén programados como entradas; mientras que la CPU automáticamente la desconectará cada vez que un terminal del puerto sea programado como salida, por medio del respectivo *bit* en el registro **DDRAx**.

La **figura 18.3** muestra el registro habilitador de las resistencias *pull-up* del puerto A (**PTAPUE**).

En la **figura 18.4** se ha dibujado la configuración interna en bloques del circuito del puerto A, allí se puede ver que, cuando el registro **DDRAx** está en uno lógico, la lectura de la dirección $0000 se efectúa sobre el registro de datos **PTAx**; pero cuando el registro **DDRAx** está en cero lógico, la lectura de dicha posición es el nivel de voltaje presente en el terminal.

El puerto B

Este puerto posee 8 terminales bidireccionales, programables, y accesibles a través del registro de datos del puerto B (**PTB**), y todos comparten su función con ocho entradas o canales del módulo de convertidores análogo-digitales (ADC).

Por medio del registro de dirección de datos del puerto B (**DDRB**), se podrá programar la condición de entrada o salida sobre cada terminal del puerto; con un uno lógico en cualquiera de sus *bits* se programa el puerto correspondiente como salida, mientras que con un cero lógico, se deshabilita el *buffer* de salida y queda programado como entrada.

Tras la acción del *reset* se afectará al registro **DDRB**, borrando todos sus *bits* de control, y

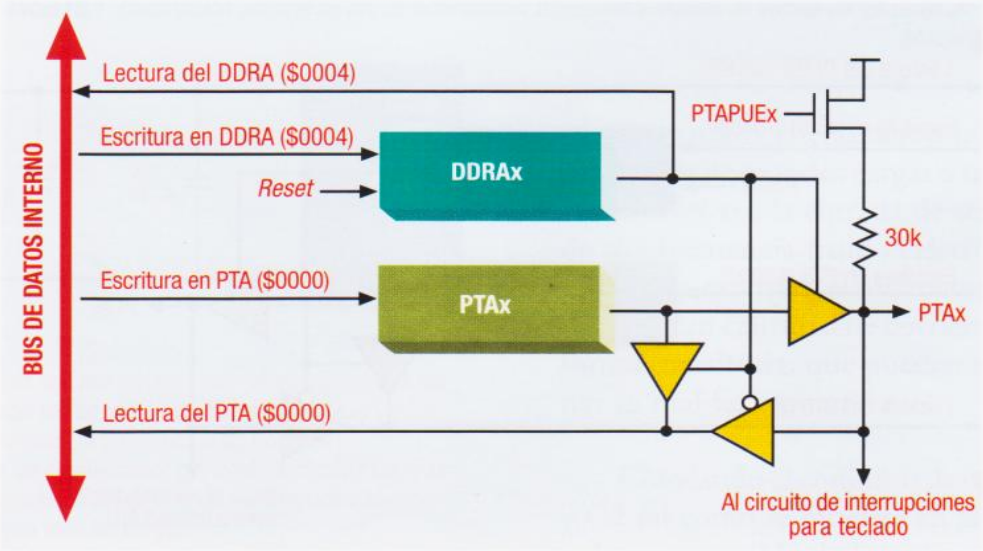


Figura 18.4 *Configuración interna en bloques del circuito del puerto A*

Figura 18.5 *Definición de los registros PTB y DDRB*

programando todo el puerto B como entrada. En la **figura 18.5** se resumen las direcciones, las características y las funciones alternas propias del registro de datos del puerto B (**PTB**) y del registro de dirección de datos B (**DDRB**).

En la **figura 18.6** se ha dibujado la configuración interna en bloques que corresponde con el puerto B; allí se puede ver que cuando el registro **DDRBx** está en uno lógico, la lectura del contenido de la dirección $0001 se efectúa sobre el *latch* del registro de datos **PTBx**; pero cuando el registro **DDRBx** está en cero lógico, la lectura de dicha posición es el nivel de voltaje presente en el terminal.

El puerto D

El puerto D está compuesto por 8 *bits* de lectura y escritura que se pueden acceder a través del registro de datos del puerto D (**PTD**), y que además poseen funciones especiales de entrada y salida digitales. Dos de sus terminales comparten su función con la interfaz del módulo temporizador (**TCH1** y **TCH0**), mientras que otros cuatro se comparten con las entradas del módulo de convertidores A/D.

En este puerto, adicionalmente, los terminales **PTD6** y **PTD7** están diseñados para manejar corrientes de hasta 25mA en configuración sumidero (*sinking*), y poseen resistencia *pull-up* programable. También los terminales **PTD2**, **PTD3**, **PTD6** y **PTD7** están capacitados para manejar directamente diodos LED.

Por medio del registro de dirección de datos del puerto D (**DDRD**), se configura la dirección del dato sobre cada terminal del puerto; con un uno lógico en cualquiera de sus *bits* se programa el puerto correspondiente como salida, mientras que con un cero lógico se deshabilita el *buffer* de salida y se programa como entrada.

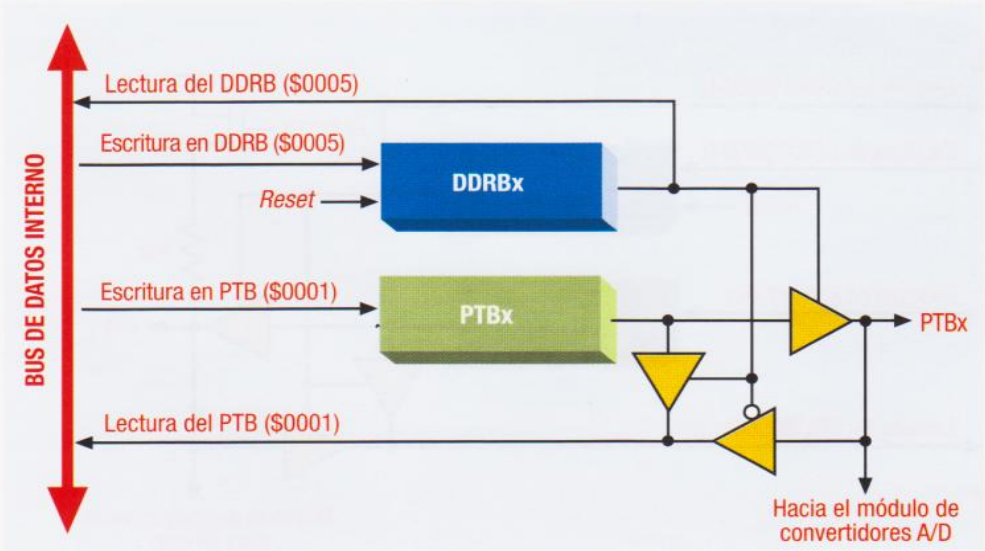


Figura 18.6 *Configuración interna en bloques del circuito del puerto B*

Dirección del registro PTD: $0003

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: / Escritura:	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
Reset:	No es afectado							
Funciones adicionales:	LED	LED			LED	LED		
					ADC8	ADC9	ADC10	ADC11
			TCH1	TCH0				
	25mA (*sink*)	25mA (*sink*)						
	5k *pull-up*	5k *pull-up*						

Dirección del registro DDRD: $0007

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: / Escritura:	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
Reset:	0	0	0	0	0	0	0	0

Figura 18.7 *Definición de los registros PTD y DDRD*

Dirección del registro PDCR: $000A

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura:	0	0	0	0	SLOWD7	SLOWD6	PTDPU7	PTDPU6
Escritura:								
Reset:	0	0	0	0	0	0	0	0

SLOWDx: ***bits* habilitadores del amplificador de corriente de 25mA**
***Bit* en 1:** habilita el amplificador
***Bit* en 0:** desconecta el amplificador
PTDPUx: ***bits* habilitadores de las resistencias pull-up de PTD6 y PTD7**
***Bit* en 1:** conecta la resistencia *pull-up* de 5k
***Bit* en 0:** desconecta la resistencia *pull-up*

Figura 18.8 *Definición del registro de control de puerto D (PDCR)*

Una vez se ejecute la acción del *reset*, el registro **DDRD** será modificado por la CPU borrando todos sus *bits* de control y configurando todo el puerto D como entrada. En la **figura 18.7** se resumen las direcciones, las características y las funciones alternas propias del registro de datos del puerto D (**PTD**) y del registro de dirección de datos D (**DDRD**).

El puerto D tiene además otro registro que nos permite habilitar las resistencias *pull-up* internas y los amplificadores de corriente de drenador abierto, asociados con los terminales **PTD6** y **PTD7**; es conocido como el registro de control del puerto D (**PDCR**). En la **figura 18.8** se muestra este registro.

En la **figura 18.9** se ha dibujado la configuración interna en bloques del circuito que corresponde al puerto D, allí se puede ver que, cuando el registro **DDRDx** está en uno lógico, la lectura de la dirección $0003 se efectúa sobre el registro de datos **PTDx**; pero cuando el registro **DDRDx** está en cero lógico, la lectura de dicha posición corresponde con el nivel de voltaje presente en el terminal.

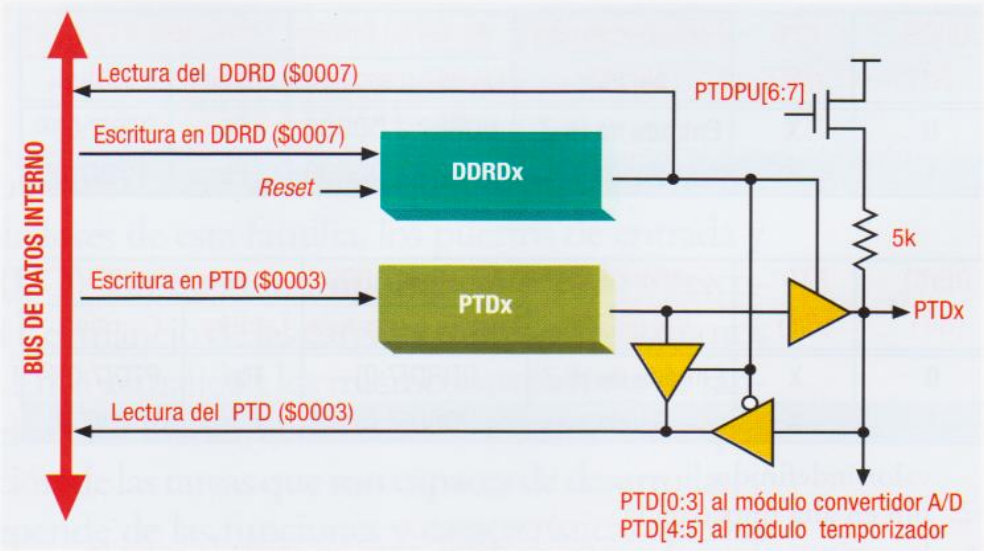


Figura 18.9 *Configuración interna en bloques del circuito del puerto D*

PARA SABER MÁS

Programación de puertos bidireccionales

Ya que todos los *pines* de los puertos de los microcontroladores Motorola pueden trabajar como entradas o salidas programables, es importante saber que el registro de datos, de cualquier puerto PTA, PTB o PTD, puede ser escrito o grabado con un nuevo dato en cualquier momento y sin importar el estado de los *bits* de control de los registros que definen la dirección de los datos en el puerto DDRA, DDRB y DDRD.

Por lo anterior, es importante que a la hora de desarrollar la rutina de programación de una tarea que involucre los puertos, el programador no olvide ubicar primero las instrucciones o rutinas que se escriben sobre el registro de datos del puerto, antes de cambiar los *bits* de control del registro de dirección de datos de 0 a 1 (de entrada a salida). De esta forma, se evita que el puerto arroje datos no esperados durante el cambio de dirección de entrada a salida que puedan afectar el comportamiento de los dispositivos periféricos externos con alta velocidad de respuesta.

El sumario de todas las características y funciones principales y compartidas de los tres registros de datos de los puertos PTA,PTB y PTD, de los microcontroladores de la familia 68HC08 de Motorola, se muestra en la **figura 18.10**.

Puerto A

PTAPUE (*Bit*)	DDRA (*Bit*)	PTA (*Bit*)	Configuración del pin I/O	Acceso al DDRA	Acceso al PTA	
				Lectura o escritura	Lectura	Escritura
1	0	X	Entrada de V_{DD}(1)	DDRA6 a DDRA0	Pin	PTA6 a PTA0(2)
0	0	X	Entrada de Hi-Z	DDRA6 a DDRA0	Pin	PTA6 a PTA0(2)
X	1	X	Salida	DDRA6 a DDRA0	PTA6 a PTA0	PTA6 a PTA0

(1): el pin de entrada o salida se sujeta a VDD por medio de la resistencia interna de *pull-up*

Puerto B

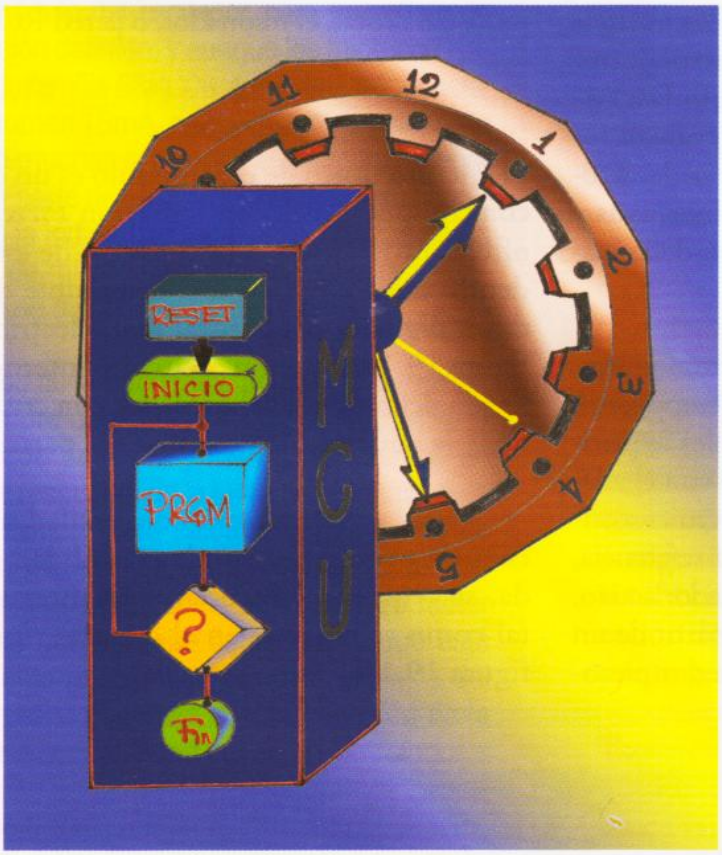
DDRB (*Bit*)	PTB (*Bit*)	Configuración del pin I/O	Acceso al DDRB	Acceso al PTB	
			Lectura o escritura	Lectura	Escritura
0	X	Entrada de Hi-Z	DDRB7 a DDRB0	Pin	PTB[7:0](2)
1	X	Salida	DDRB7 a DDRB0	Pin	PTB[7:0]

Puerto D

DDRD (*Bit*)	PTD (*Bit*)	Configuración del pin I/O	Acceso al DDRD	Acceso al PTD	
			Lectura o escritura	Lectura	Escritura
0	X	Entrada de Hi-Z	DDRD[7:0]	Pin	PTD[7:0](2)
1	X	Salida	DDRD[7:0]	Pin	PTD[7:0]

X: valor indefinido
Hi-Z: pin en alta impedancia
(2): la escritura afecta al registro de datos, pero no afecta la entrada

Figura 18.10 *Resumen de todas las características y funciones definidas para los puertos A, B y D*



EL OSCILADOR Y EL *RESET*, EN LOS MICROCONTROLADORES HC08 DE MOTOROLA

La ejecución de la secuencia correcta de las tareas programadas y la mayoría de funciones que se llevan a cabo en los módulos internos del microcontrolador, se desarrollan de acuerdo a cada uno de los periodos de la señal de reloj generada desde el módulo del oscilador, a partir de componentes externos.

El *reset* es la señal que goza de la mayor jerarquía dentro del conjunto de interrupciones disponibles en los microcontroladores de cualquier familia, ella le define al programador un lugar para el inicio de la ejecución del código del programa.

El módulo interno del oscilador (OSC)

Si lugar a dudas, uno de los módulos periféricos internos de mayor importancia es el oscilador, éste genera las señales de reloj que definen la velocidad de ejecución del programa por parte del dispositivo, y en ellas se basan todas las cuentas de tiempo que se hacen dentro del microcontrolador. Según el periodo de oscilación (o fase) de esta señal, se sincronizan, ejecutan, y evalúan todas las tareas que en el interior de éste se desarrollan; tales como: los ciclos básicos de búsqueda y captura, de ejecución o decodificación de la instrucción, de lectura, y de escritura.

El módulo del oscilador está construido internamente con los circuitos, registros y compuertas necesarias para generar las señales de reloj requeridas por la CPU con gran estabilidad, y con el adecuado nivel de voltaje y frecuencia. Los microcontroladores Motorola en función de su referencia, poseen uno de los dos sistemas de oscilador activo, uno de ellos genera la señal de reloj a partir de un cristal de cuarzo **XTAL** o de un resonador piezoeléctrico; mientras que el otro lo hace por medio de una red conformada por una resistencia y un condensador **RC**; y los dos sistemas comparten las mismas entradas externas (*pines* **OSC1** y **OSC2**) asignadas en el microcontrolador para la conexión externa del cristal, el resonador, o la red RC.

Módulo oscilador con cristal XTAL (subfamilia 68HC08xxx)

El circuito típico de este módulo es un oscilador con compuertas en configuración ***Pierce***, el cual ofrece muy buenas características de estabilidad dentro de la amplia gama disponible de osciladores digitales para cristal piezoeléctrico. En la **figura 19.1** se muestra el circuito interno simplificado del módulo del oscilador con cristal externo, y la conexión lógica entre sus partes.

Para la configuración del oscilador a la frecuencia necesaria y con la estabilidad apropiada, se requieren cinco componentes externos, tal como se muestra en el circuito típico de la figura 19.1, y éstos son:

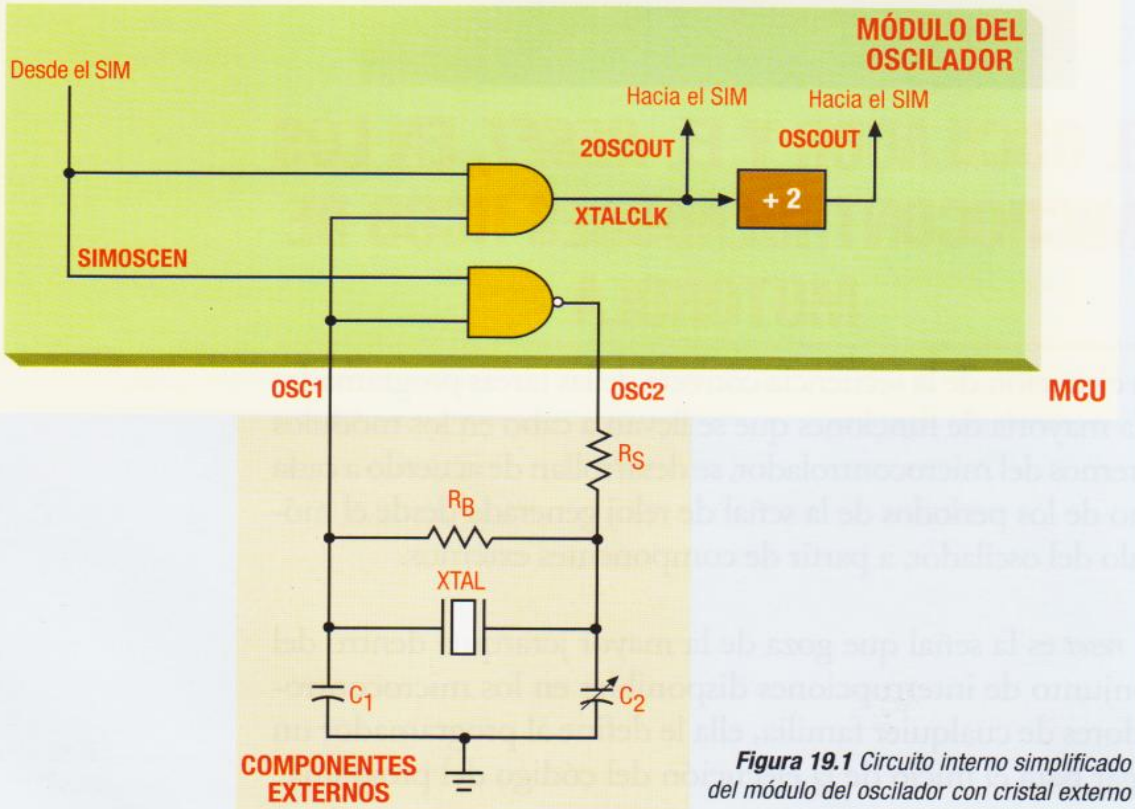


Figura 19.1 *Circuito interno simplificado del módulo del oscilador con cristal externo*

1. **El cristal de cuarzo o el resonador piezoeléctrico XTAL**, éste puede tener una frecuencia de resonancia máxima de 32MHz, y una capacitancia equivalente CL entre 5 y 40pF. Su conexión con el módulo interno del microcontrolador se hace a través de los terminales OSC1 y OSC2.

2. **El condensador fijo C1**, éste debe tener una estrecha tolerancia (menor al 5%), y su valor típico debe ser alrededor de dos veces la capacitancia equivalente del cristal o del resonador. Su conexión con el microcontrolador se hace por medio de la entrada OSC1, que es la entrada del oscilador interno.

3. **El condensador variable o de sintonía C2**, éste debe tener una estrecha tolerancia (menor al 5%), y de la misma forma que para C1, su valor típico debe ser alrededor de dos veces la capacitancia equivalente del cristal o del resonador. Usando un condensador ajustable de baja tolerancia en el lugar de C2, se podrá sintonizar cualquier frecuencia no estándar de trabajo inferior a la de resonancia del cristal, para los desarrollos que así lo requieran; el condensador de sintonía se debe conectar al circuito del oscilador en su salida, o sea a través del terminal OSC2.

4. **La resistencia de realimentación RB**, ésta puede tener una tolerancia de hasta el 10% y su valor típico es 10MΩ; su función consiste en facilitar el arranque de la oscilación con cualquier cristal o resonador, una vez que el circuito es alimentado y habilitado desde la CPU. Su ubicación es en el lazo de realimentación, entre la entrada y la salida del módulo oscilador, o sea entre los terminales OSC1 y OSC2.

5. **La resistencia *Pierce* para baja frecuencia Rs**, ésta puede tener una tolerancia de hasta el 10% y sólo se requiere cuando el oscilador va a trabajar a una frecuencia baja (inferior a 1MHz); tal como se requiere en los diseños estándar de los osciladores digitales en configuración *Pierce*. Si no se utiliza, se sustituye por un corto circuito.

Módulo oscilador con red RC (subfamilia 68HRC08xxx)

El circuito oscilador de este módulo usa una red externa conformada por un condensador y una resistencia, y circuitos internos que controlan y convierten la señal de carga y descarga del condensador en un tren de pulsos rectangulares, sin distorsión, y relativamente estables.

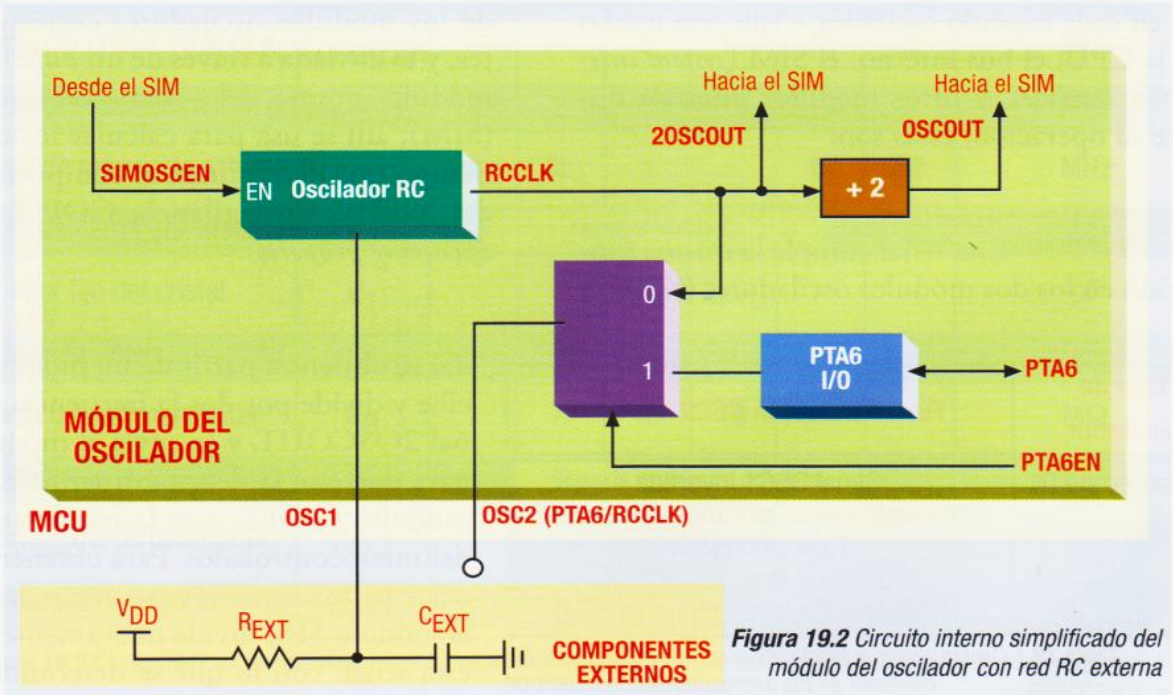


Figura 19.2 Circuito interno simplificado del módulo del oscilador con red RC externa

En términos de estabilidad y precisión, los osciladores cuya frecuencia de salida depende de una red con una resistencia y un condensador (RC), siempre estarán sujetos a la calidad de dicha red; en el módulo del oscilador RC de esta subfamilia de microcontroladores se requiere que los componentes externos sean de precisión (con tolerancia inferior o igual a 1%) para obtener una señal de reloj con una tolerancia del 10% o menos. La conexión de la red RC, con el módulo interno del oscilador, se hace a través del terminal OSC1; mientras que en el terminal OSC2 queda habilitado el puerto bidireccional PTA6 o disponible la señal de salida de reloj del oscilador RCCLK.

En la **figura 19.2** se muestra el circuito interno simplificado del módulo del oscilador con red RC externa, y la conexión lógica entre sus partes.

En la **tabla 19.1** se resume el comportamiento de los terminales disponibles en el microcontrolador para la configuración del módulo del oscilador.

Señales de reloj internas

En las figuras 19.1 y 19.2 se pueden ver también las principales señales que intervienen y se originan en cada módulo oscilador, y que son usadas por la CPU, el bus interno, el SIM (*system integration module*), y otros módulos internos durante su operación; éstas son:

1. **La señal de habilitación del oscilador, SIMOSCEN.** Esta señal cumple la misma función en los dos módulos osciladores (XTAL y RC); por medio de ella el módulo del sistema de integración SIM del microcontrolador, habilita o inhabilita completamente el funcionamiento de cualquiera de los osciladores.

2. **Salida de reloj del oscilador con cristal, XTALCLK.** Esta señal es la salida de reloj del módulo oscilador con cristal y su periodo está dado típicamente por la frecuencia nominal del cristal de cuarzo (F_{XCLK}); el ciclo de servicio de la señal dependerá de la calidad y los parámetros de fabricación del cristal, y no se debe olvidar que su frecuencia y amplitud serán inestables durante el breve periodo de tiempo que dura el ciclo de arranque (*start-up*) del microcontrolador, a partir de la conexión a la alimentación.

3. **Salida de reloj del oscilador con red RC, RCCLK.** Ésta es la señal de salida del módulo oscilador RC y su frecuencia y estabilidad son directamente proporcionales a la constante de tiempo de la red externa RC, y a la precisión y calidad de sus componentes respectivamente.

4. **Salida del oscilador 2OSCOUT.** Ésta es la misma señal de salida XTALCLK ó RCCLK de los módulos osciladores correspondientes, y es enviada a través de un *buffer* hacia el módulo interno del sistema de integración (SIM), allí se usa para calcular los ciclos de tiempo que determinan el comportamiento del módulo de vigilancia COP (*computer operating properly*).

5. **Salida del oscilador, OSCOUT.** Esta señal se obtiene a partir de un bloque que recibe y divide por dos la frecuencia de la señal 2OSCOUT, y se envía al módulo SIM para que éste la distribuya entre los demás módulos internos que la requieran dentro del microcontrolador. Para obtener la señal del reloj, usada por el bus interno de la CPU, el módulo SIM divide nuevamente por dos esta señal, con lo que se determina que la

Tipo de oscilador	Función del pin OSC2
Oscilador con XTAL	Señal OSC1 invertida
Oscilador con RC	Controlado por el *bit* PTAEN del registro PTAPUER ($0D) PTA6EN = 0: salida RCCLK PTA6EN = 1: Línea de puerto PTA6

Tabla 19.1 *Configuración y funciones asignadas a los* pines *del oscilador en los microcontroladores 68HC08*

frecuencia del reloj del bus interno de la CPU (F_{OP}) sea siempre la cuarta parte de la frecuencia de la señal del oscilador RC o XTAL (XTALCLK ó RCCLK).

Características eléctricas de los módulos osciladores

Aunque los microcontroladores Motorola cuentan con una amplia tolerancia para el voltaje de alimentación (entre 3 y 5V), esto no quiere decir que muchas de sus características eléctricas fundamentales para su operación no sean afectadas con los cambios, tal como sucede con la frecuencia en el módulo oscilador.

La frecuencia de operación de todas las señales de reloj manejadas por el microcontrolador, es especialmente afectada por los cambios en la magnitud de la tensión de alimentación, e igualmente, el consumo total de corriente de éste (I_{DD}) es función directa de la frecuencia de operación que se elija.

La **tabla 19.2** es un sumario de las magnitudes típicas que se deben considerar a la hora de configurar los módulos de oscilación de los microcontroladores Motorola 68HC08 y 68HRC08; la tabla se muestra para los dos voltajes nominales de alimentación estandarizados: 5 y 3VCC.

Para calcular los componentes externos de la red RC, se recomienda usar un condensador externo de 10pF con baja tolerancia, elegir una frecuencia, y calcular la resistencia por medio de una de las dos curvas que se muestran en la **figura 19.3**; en la figura se muestran las curvas de resistencia contra frecuencia, que corresponden con dos niveles de alimentación típicos, allí se puede ver con claridad la influencia del voltaje de alimentación en la frecuencia del reloj.

El *RESET* en los microcontroladores Motorola 68HC08

Dentro de la gama de interrupciones internas y externas, el *reset* o interrupción de inicialización

Características	Símbolo	Mínimo		Típico		Máximo		Unidad
		3V	5V	3V	5V	3V	5V	
Frecuencia de operación interna	F_{OP}	-	-	-	-	4	8	MHz
Pulso bajo de reset mínimo ($\overline{RST}$)	T_{IRL}	1.5	750	-	-	-	-	µs, ns
Frecuencia del cristal, XTALCLK	$F_{OSCXCLK}$	-	-	8	10	16	32	MHz
Frecuencia del oscilador, RCCLK	F_{RCCLK}	2	2	8	10	12	12	MHz
Frecuencia de referencia externa	$F_{OSCXCLK}$	dc	dc	-	-	16	32	MHz
Capacidad de carga del cristal	C_L	Según el fabricante						pF
Condensador fijo del cristal	C_1	-	-	$2xC_L$	$2xC_L$	-	-	pF
Condensador de sintonía del cristal	C_2	-	-	$2xC_L$	$2xC_L$	-	-	pF
Resistencia de realimentación	R_B	-	-	10	10	-	-	MΩ
Resistencia en serie del oscilador	R_S	Según la impedancia del cristal						kΩ
Resistencia externa del oscilador RC	R_{EXT}	Calcular según la figura 19.3						kΩ
Condensador externo del oscilador RC	C_{EXT}	-	-	10	10	-	-	pF

Tabla 19.2 *Características y especificaciones de componentes externos para los módulos osciladores de los microcontroladores 68HC08 y 68HRC08.*

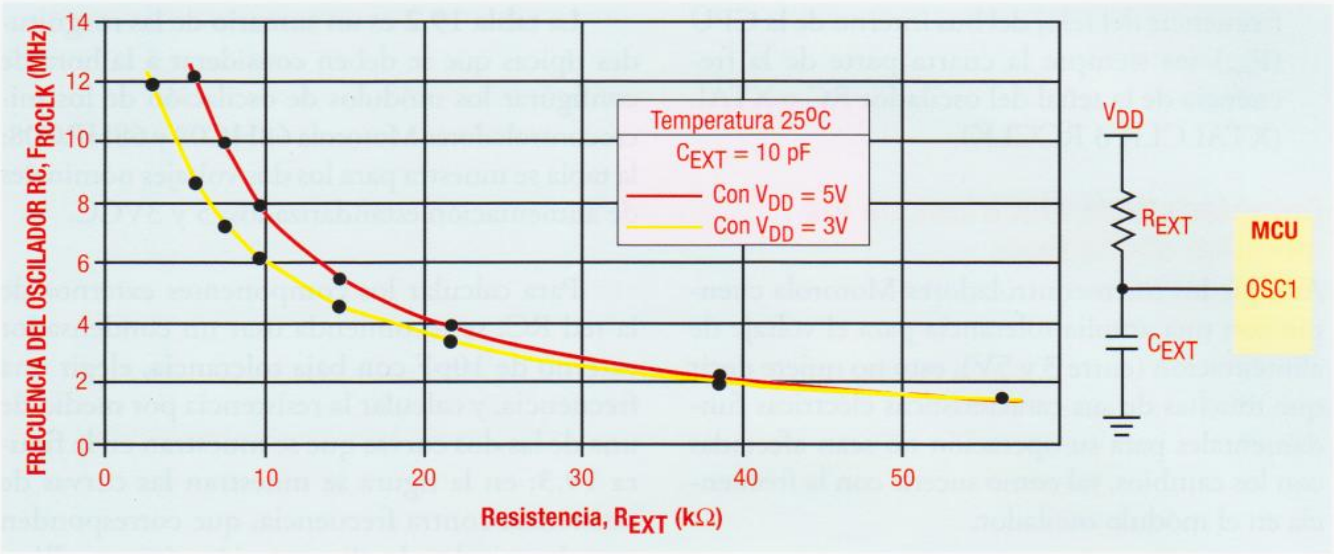


Figura 19.3 *Curvas de resistencia externa contra frecuencia del oscilador RC, para el cálculo de Rext*

del microcontrolador es la fuente de mayor jerarquía, esto significa que su petición será atendida inmediatamente luego de su reconocimiento y validación.

Cada vez que la CPU del microcontrolador recibe y valida una señal de interrupción de *reset*, el microcontrolador es obligado a asumir un conjunto de condiciones iniciales para sus módulos internos, y a iniciar la búsqueda y ejecución de las instrucciones en un lugar predeterminado por los fabricantes para el inicio (*starting address*).

La entrada de *reset* no es sincrónica con el ciclo de ejecución de instrucciones, y por lo tanto la interrupción será atendida en cualquier momento, inmediatamente después de recibir un nivel de voltaje en nivel bajo por el *pin* de *reset*; en la **figura 19.4** se aprecia el diagrama de tiempos que representa la ejecución de un ciclo de *reset* al interior del microcontrolador.

Condiciones iniciales establecidas por medio del ciclo de *reset*

Una vez la condición de *reset* externa o interna se reconoce, inmediatamente muchos registros y *bits* de control son grabados con los datos necesarios para iniciar la rutina de trabajo con el microcontrolador, a partir de una posición y con unos parámetros definidos; muchas de estas con-

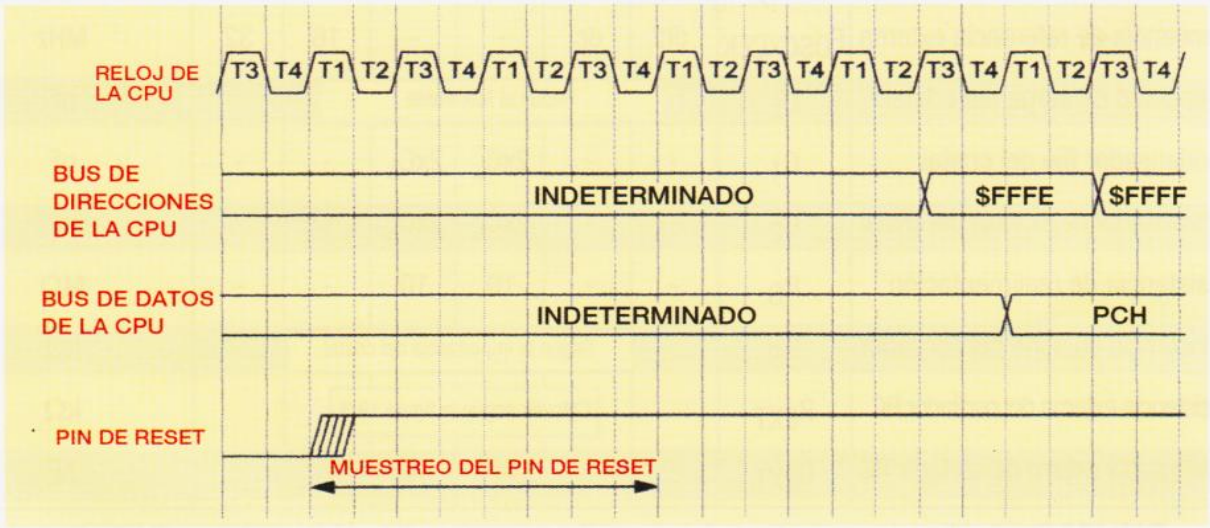


Figura 19.4 *Diagrama de tiempos de un ciclo de* reset *activado externamente por medio de la correspondiente entrada*

PARA SABER MÁS

El reloj y la potencia del microcontrolador

En todos los microcontroladores Motorola, la potencia consumida depende directamente de la frecuencia de operación a la que haya sido configurado el módulo del oscilador, para un voltaje de alimentación definido; y aunque los microcontrladores de la familia 68HC08, a máxima frecuencia, sólo consumen 15mA, es importante considerarlo ya que en muchas ocasiones el microcontrolador puede ser parte de un gran sistema, que habitualmente en la industria puede contener más de un centenar de microcontroladores y poseer limitaciónes en su disponibilidad de energía.

La elección de la frecuencia adecuada de operación, siempre debe ser un compromiso entre el consumo de potencia y los requerimientos en tiempo de respuesta de la tarea por ejecutar, ésta se debe evaluar para calcular la frecuencia de operación mínima y luego configurar el oscilador del microcontrolador a tal valor. Otro factor importante que se debe considerar y que está directamente relacionado con la frecuencia del oscilador, es el desgaste o la disminución de la vida útil y la confiabilidad del microcontrolador, a causa de un aumento en el consumo de potencia que se traduce en un incremento de la temperatura del circuito integrado.

En la figura se han dibujado las curvas que relacionan el consumo de la corriente de alimentación (I_{DD}) y la frecuencia del bus interno de la CPU (F_{OP}) en los microcontroladores 68HC08, para los dos voltajes de alimentación típicos.

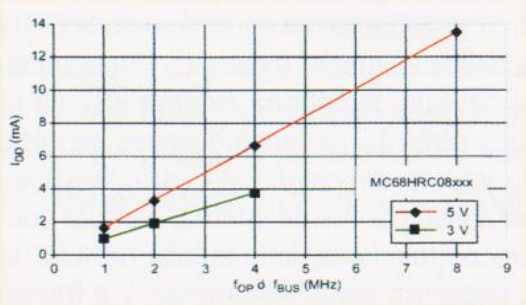


Figura 19.5 *Comportamiento de la corriente de alimentación del microcontrolador con todos sus módulos internos activos (I_{DD}), frente a la frecuencia de operación del reloj del bus interno de la CPU (F_{OP})*

diciones iniciales son independientes del modo de operación, y algunas están destinadas a permitir el arranque apropiado de algunos módulos internos fundamentales para la CPU; mientras que otras, definen puntos de referencia entre el programador y la CPU que deben considerarse para ejecutar correctamente el código de operación de los programas.

Las principales acciones que se desarrollan, una vez que el microcontrolador es inicializado a través del *reset*, se describen a continuación.

1. Inicialización de la CPU08

Después de la ejecución del ciclo de *reset*, el registro **PC** es grabado con las posiciones de memoria **$FFFE** y **$FFFF**, definidas para el respectivo vector (un vector representa el contenido de una localidad de memoria definida y asignada a un evento externo o interno), y la CPU inicia el ciclo de ejecución o decodificación de las instrucciones allí contenidas, que por lo general son instrucciones de salto.

También el registro **SP** se graba con el dato **$00FF**, ubicando la pila en la primera página de la memoria de datos, mientras que el registro índice **H** es borrado a cero lógico con el objetivo de hacer compatible la migración del código de programa entre las familias HC05 y HC08.

Todos los demás registros de la CPU asumen un estado indeterminado luego del *reset*. Sin embargo, algunos *bits* de control del registro del código de condiciones **CCR**, tales como la máscara de interrupciones I, y las banderas de los modos de operación **STOP** y **WAIT**, son puestos en uno y cero lógico respectivamente, para inhabilitar todas las fuentes de interrupción y llevar la CPU al modo normal de trabajo.

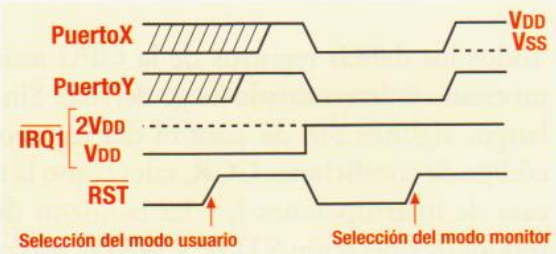
2. Definición del modo de operación de la CPU08

En los microcontroladores Motorola, la CPU08 cuenta con dos modos de operación

disponibles para el programador: el modo usuario y el modo monitor; en el modo monitor la CPU tiene el mismo comportamiento que en el modo usuario, excepto que la dirección definida para el vector del *reset* se cambia por las posiciones reservadas **$FEFE** y **$FEFF**.

Con este desplazamiento la CPU inicia los ciclos de búsqueda y ejecución del código desde el *firmware* interno del módulo monitor y no directamente en el código del programador.

El modo de operación se define y es memorizado por la CPU durante el flanco de subida del voltaje presente en el *pin* de *reset* (durante la transición de V_{SS} a V_{DD}); el modo usuario es el modo de trabajo definido por defecto tras la inicialización del microcontrolador, pero la selección del modo monitor se efectúa llevando dos de sus líneas de puertos I/O de propósito general a V_{SS} y aplicando en la entrada de interrupción $\overline{IRQ1}$ un sobrevoltaje igual al doble de V_{DD}; en la **figura 19.6** se muestran gráficamente y se resumen en un cuadro las condiciones para seleccionar el modo de operación.



Pin $\overline{IRQ1}$	Puerto X	Puerto Y	Modo
$\leq V_{DD}$	X	X	Usuario
$2 \times V_{DD}$	0	0	Monitor

Figura 19.6 *Condiciones para la selección del modo de operación de la CPU08*

Fuentes de interrupción del *reset*

Todas las fuentes de interrupción del *reset* detienen inmediatamente la ejecución de cualquier instrucción en curso dentro del microcontrolador, vectorizan hacia las posiciónes $FFFE y $FFFF, inician un ciclo de *reset* interno que graba todos los registros de la CPU con sus valores por defecto, y restablecen las condiciones iniciales en todos los módulos periféricos internos.

El ciclo de *reset* se puede activar desde diversas fuentes internas y externas al microcontrolador, como son: el *reset* externo (*pin* $\overline{RST}$), el *reset* a la conexión (*power-on reset* POR), el *reset* del temporizador *watchdog* COP, el *reset* por código de operación ilegal, el *reset* por dirección ilegal y la interrupción de inhibición por bajo voltaje LVI.

PARA SABER MÁS
Validación del *reset* externo e interno

Para que el ciclo de *reset* sea activado, las señales que lo inician deben cumplir ciertas características de tiempo que dependen del origen de la fuente de interrupción. Cuando la fuente es externa, en el *pin* de *reset* ($\overline{RST}$) se debe aplicar un nivel lógico cero (V_{SS}) durante un tiempo mínimo de 750ns, cuando V_{DD} es 5V, ó de 1,5µs, cuando V_{DD} es 3V, para que el módulo interno del *reset* lo reconozca, valide e inicie las acciones propias de dicho ciclo.

En el caso de las fuentes internas, su reconocimiento y validación se inicia cuando la CPU coloca desde el interior y durante 16 periodos del reloj del bus, la entrada externa $\overline{RST}$ en nivel cero, y otros 16 periodos después se inicia el ciclo interno de *reset*. Esta característica del *reset* activo por fuente interna, permite que los microcontroladores de la familia 68HC08 puedan inicializar simultáneamente, y a través de dicha entrada, otros circuitos integrados, periféricos externos, y microcontroladores esclavos, inmersos en un sistema complejo.

LECCIÓN 20

ORGANIZACIÓN DE LA MEMORIA EN LOS MICROCONTROLADORES HC08 DE MOTOROLA

En esta ocasión se estudiará el mapa de memoria de los microcontroladores Motorola de la familia 68H(R)C08. En él, se muestra la distribución de la memoria disponible y direccionable la cual está conformada por la memoria de datos RAM, la de programación ROM, y el área de memoria ROM del programa monitor.

Una característica particular de los microcontroladores de esta familia, es la disponibilidad para el usuario de un programa monitor residente en la memoria (*firmware*), que facilita la ejecución y depuración de programas a través del puerto serial de una PC.

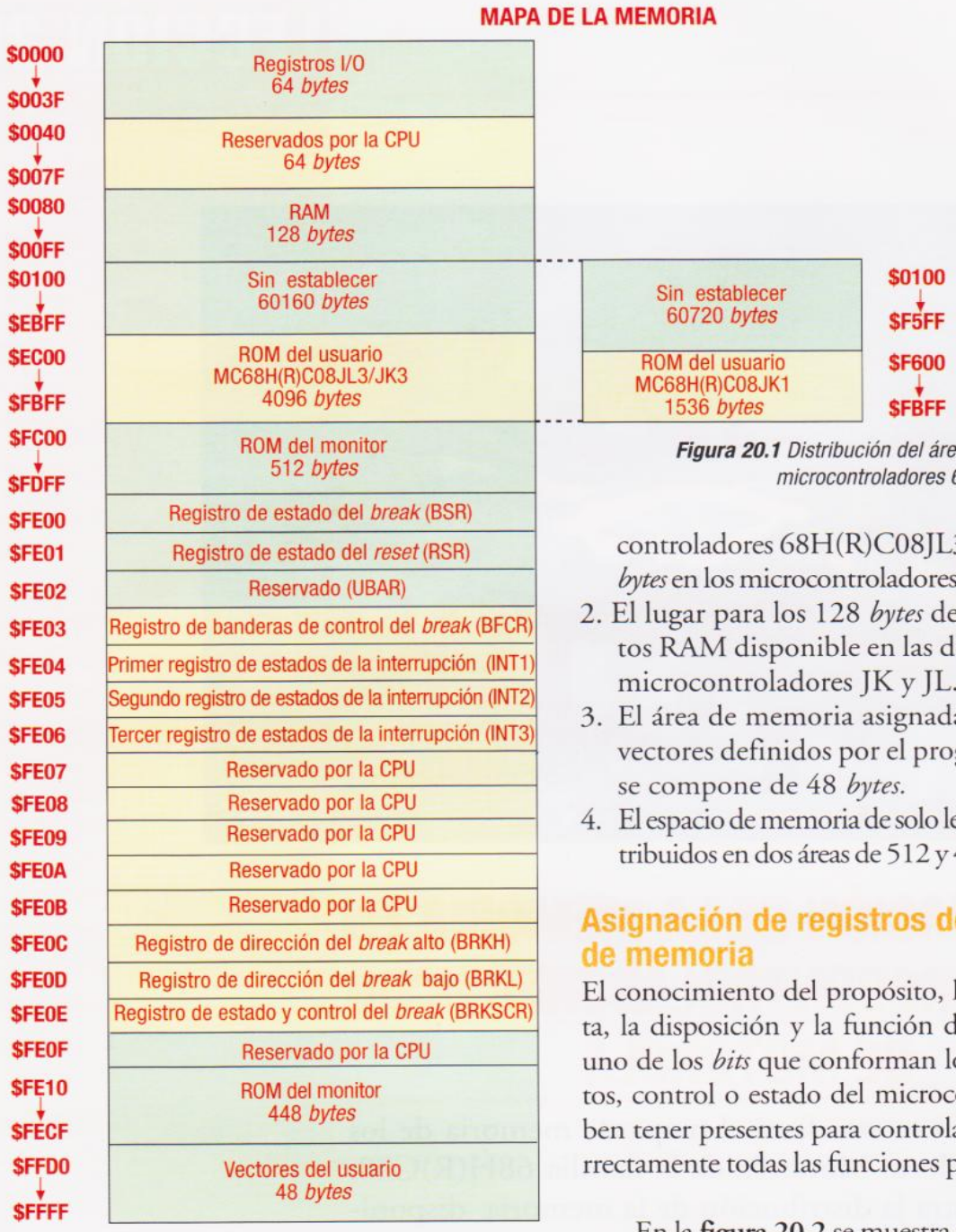


Figura 20.1 *Distribución del área de la memoria en los microcontroladores 68H(R)C08JK1/JK3/JL3*

Organización de la memoria

La CPU08 de esta familia de microcontroladores posee un bus de direcciones de 16 *bits* con el que se pueden acceder hasta 64K*bytes* del área de la memoria. El mapa de la memoria que se puede apreciar en la **figura 20.1**, muestra su organización, y en él se incluyen las siguientes áreas:

1. El espacio de memoria ROM disponible para el usuario, que es de 4.096 *bytes* para los microcontroladores 68H(R)C08JL3/JK3, y de 1.536 *bytes* en los microcontroladores 68H(R)C08JK1.
2. El lugar para los 128 *bytes* de memoria de datos RAM disponible en las dos subfamilias de microcontroladores JK y JL.
3. El área de memoria asignada para la tabla de vectores definidos por el programador, la cual se compone de 48 *bytes*.
4. El espacio de memoria de solo lectura (ROM), distribuidos en dos áreas de 512 y 448 *bytes* cada una.

Asignación de registros dentro del área de memoria

El conocimiento del propósito, la ubicación exacta, la disposición y la función definida para cada uno de los *bits* que conforman los registros de datos, control o estado del microcontrolador, se deben tener presentes para controlar y supervisar correctamente todas las funciones programadas en él.

En la **figura 20.2** se muestra en detalle la distribución y asignación que se le ha dado al espacio de memoria direccionable por el microcontrolador, contenido entre las direcciones $0000 y $003F; en él se encuentran contenidos la gran mayoría de registros de control, de estado y de datos. Adicionalmente, algunos registros reservados por la CPU y de propósito especial, se han ubicado en las últimas16 posiciones de memoria localizadas entre $FE00 y $FEOE y la última posición $FFFF que corresponde al registro de control del COP (COPCTL).

REGISTROS DE PROPÓSITO ESPECIAL

Dirección	Nombre del registro		Bit 7	6	5	4	3	2	1	Bit 0
$0000	Registro de datos del puerto A (PTA)	Lectura:	0	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0
		Escritura:								
		Reset:	No se afecta por el *reset*							
$0001	Registro de datos del puerto B (PTB)	Lectura:	PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0
		Escritura:								
		Reset:	No se afecta por el *reset*							
$0002	Sin establecer	Lectura:								
		Escritura:								
$0003	Registro de datos del puerto D (PTD)	Lectura:	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
		Escritura:								
		Reset:	No se afecta por el *reset*							
$0004	Registro de dirección de datos A (DDRA)	Lectura:	0	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$0005	Registro de dirección de datos B (DDRB)	Lectura:	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$0006	Sin establecer	Lectura:								
		Escritura:								
$0007	Registro de dirección de datos D (DDRD)	Lectura:	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$0008 ↓ $0009	Sin establecer	Lectura:								
		Escritura:								
$000A	Registro de control del puerto D (PDCR)	Lectura:	0	0	0	0	SLOWD7	SLOWD6	PTDPU7	PTDPU6
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$000B ↓ $000C	Sin establecer	Lectura:								
		Escritura:								
$000D	Registro habilitador de entradas *pull-up* del puerto A (PTAPUE)	Lectura:	PTA6EN	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$000E ↓ $0019	Sin establecer	Lectura:								
		Escritura:								
$001A	Registro de control y estado del teclado (KBSCR)	Lectura:	0	0	0	0	KEYF	0	IMASKK	MODEK
		Escritura:						ACKK		
		Reset:	0	0	0	0	0	0	0	0
$001B	Registro habilitador de interrupciones del teclado (KBIER)	Lectura:	0	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0

Dirección	Nombre del registro		Bit 7	6	5	4	3	2	1	Bit 0
$001C	Sin establecer	Lectura:								
		Escritura:								
$001D	Registro de estado y control de IRQ (INTSCR)	Lectura:	0	0	0	0	IRQF1	0	IMASK1	MODE1
		Escritura:						ACK1		
		Reset:	0	0	0	0	0	0	0	0
$001E	Registro de configuración 2 (CONFIG2)*	Lectura: / Escritura:	IRQPUD	R	R	LVIT1	LVIT0	R	R	R
		Reset:	0	0	0	0**	0**	0	0	0
$001F	Registro de configuración 1 (CONFIG1)*	Lectura: / Escritura:	COPRS	R	R	LVID	R	SSREC	STOP	COPD
		Reset:	0	0	0	0	0	0	0	0

* Después de cada reset sólo se puede escribir una vez. ** Solamente por medio del reset a la conexión (POR) las interrupciones LVIT1 y LVIT0 se colocan en 0 lógico

$0020	Registro de estado y control del TIM (TSC)	Lectura:	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
		Escritura:	0			TRST				
		Reset:	0	0	1	0	0	0	0	0
$0021	Registro alto del contador del TIM (TCNTH)	Lectura:	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$0022	Registro bajo del contador del TIM (TCNTL)	Lectura:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$0023	Registro alto del módulo del contador del TIM (TMODH)	Lectura: / Escritura:	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
		Reset:	1	1	1	1	1	1	1	1
$0024	Registro bajo del módulo del contador del TIM (TMODL)	Lectura: / Escritura:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Reset:	1	1	1	1	1	1	1	1
$0025	Registro de estado y control del canal 0 del TIM (TSC0)	Lectura:	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
		Escritura:	0							
		Reset:	0	0	0	0	0	0	0	0
$0026	Registro alto del canal 0 del TIM (TCH0H)	Lectura: / Escritura:	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
		Reset:	Indeterminado después del *reset*							
$0027	Registro bajo del canal 0 del TIM (TCH0L)	Lectura: / Escritura:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Reset:	Indeterminado después del *reset*							
$0028	Registro de estado y control del canal 1 del TIM (TSC1)	Lectura:	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
		Escritura:	0							
		Reset:	0	0	0	0	0	0	0	0
$0029	Registro alto del canal 1 del TIM (TCH1H)	Lectura: / Escritura:	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
		Reset:	Indeterminado después del *reset*							
$002A	Registro bajo del canal 1 del TIM (TCH1L)	Lectura: / Escritura:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Reset:	Indeterminado después del *reset*							

Dirección	Nombre del registro		Bit 7	6	5	4	3	2	1	Bit 0
$002B ↓ $003B	Sin establecer	Lectura:								
		Escritura:								
$003C	Registro de estado y control del ADC (ADSCR)	Lectura:	COCO	AIEN	ADCO	CH4	CH3	CH2	CH1	CH0
		Escritura:								
		Reset:	0	0	0	1	1	1	1	1
$003D	Registro de datos del ADC (ADR)	Lectura:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
		Escritura:								
		Reset:	Indeterminado después del *reset*							
$003E	Registro de entrada de reloj del ADC (ADICLK)	Lectura:	ADIV2	ADIV1	ADIV0	0	0	0	0	0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$003F	Sin establecer	Lectura:								
		Escritura:								
$FE00	Registro de estado del *break* (BSR)	Lectura:	R	R	R	R	R	R	SBSW	R
		Escritura:							Ver nota	
		Reset:							0	
			NOTA: Escribiendo un 0 lógico se borra el *bit* SBSW							
$FE01	Registro de estado del *reset* (RSR)	Lectura:	POR	PIN	COP	ILOP	ILAD	MODRST	LVI	0
		Escritura:								
		Reset:	1	0	0	0	0	0	0	0
$FE02	Reservado	Lectura:	R	R	R	R	R	R	R	R
		Escritura:								
$FE03	Registro de las banderas de control del *break* (BFCR)	Lectura:	BCFE	R	R	R	R	R	R	R
		Escritura:								
		Reset:	0							
$FE04	Primer registro de estado de las interrupciones (INT1)	Lectura:	0	IF5	IF4	IF3	0	IF1	0	0
		Escritura:	R	R	R	R	R	R	R	R
		Reset:	1	0	0	0	0	0	0	0
$FE05	Segundo registro de estado de las interrupciones (INT2)	Lectura:	IF14	0	0	0	0	0	0	0
		Escritura:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$FE06	Tercer registro de estado de las interrupciones (INT3)	Lectura:	0	0	0	0	0	0	0	IF15
		Escritura:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$FE07 ↓ $FE0B	Reservado	Lectura:	R	R	R	R	R	R	R	R
		Escritura:								
$FE0C	Registro de dirección del *break* alto (BRKH)	Lectura:	*Bit* 15	*Bit* 14	*Bit* 13	*Bit* 12	*Bit* 11	*Bit* 10	*Bit* 9	*Bit* 8
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$FE0D	Registro de dirección del *break* bajo (BRKL)	Lectura:	*Bit* 7	*Bit* 6	*Bit* 5	*Bit* 4	*Bit* 3	*Bit* 2	*Bit* 1	*Bit* 0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0

Dirección	Nombre del registro		Bit 7	6	5	4	3	2	1	Bit 0
$FE0E	Registro de estado y control del *break* (BRKSCR)	Lectura:	BRKE	BRKA	0	0	0	0	0	0
		Escritura:								
		Reset:	0	0	0	0	0	0	0	0
$FFFF	Registro de control del COP (COPCTL)	Lectura:	*Byte* bajo del vector del reset							
		Escritura:	La escritura de cualquier valor borra el contador del COP							
		Reset:	No es afectado por el *reset*							

Sin establecer R Reservado

Figura 20.2 *Localización y asignación de los registros y bits de control, estado y datos, que conforman el mapa de la memoria de los microcontroladores 68H(R)C08JK1/JK3/JL3*

Tabla de vectores

En la **tabla 20.1** se muestran los 16 vectores definidos para las interrupciones en los microcontroladores de esta familia; su función, su dirección exacta, y su nivel de prioridad son de gran importancia a la hora de atender los llamados de interrupción internos y externos solicitados por la CPU; estos vectores se encuentran en 48 posiciones de memoria de 16 *bits* cada una. En la tabla se puede ver también la prioridad de cada vector la cual es el parámetro que define el orden de atención que la CPU le dará a cada una de ellas en los casos cuando su activación pueda llegar a ser simultánea ante cualquier evento interno o externo.

Memoria de datos de lectura y escritura RAM

Las direcciones $0080 y $00FF definen el espacio de memoria RAM con una longitud de palabra de 8 *bits*; en ella se albergan todos los datos temporales que sean requeridos por el programador, y que requieran ser leídos y modificados en cualquier momento.

Dentro de esta área se debe ubicar el espacio para la pila de datos (*stack*), y aunque su ubicación puede ser programada y posea un registro SP de 16 *bits* que le da movilidad dentro de los 64 *Kbyte* del área de memoria, su ubicación siempre debe estar dentro del espacio RAM, y especialmente en la página cero para almacenar y recuperar eficazmente los registros de la CPU durante los llamados, saltos, e interrupciones. La página cero de la RAM está conformada por los primeros 128 *bytes*; ésta provee al usuario de un espacio ideal para el acceso frecuente de variables globales del programa y un acceso rápido a las instrucciones por medio de direccionamiento directo.

Memoria de programa de solo lectura ROM

En esta área de memoria de 8 *bits* definida entre las direcciones $EC00 y $FBFF (4096 *bytes*) para los microcontroladores 68H(R)C08JK3/JL3, o entre las direcciones $F600 y $FBFF (1536 *bytes*) para los 68H(R)C08JK1, el usuario puede guardar los datos que hacen parte del código del programa, así como las constantes, asignaciones y etiquetas que se deben preservar dentro de la memoria por tiempo indefinido y sin modificación.

En esta área también están contenidas la tabla de vectores de las interrupciones y el vector del *reset* del microcontrolador, entre las direcciones $FFD0 y $FFFF, tal como se aprecia en la tabla 20.1.

Memoria ROM del programa monitor (MON)

El programa monitor, inalterable y residente en la memoria ROM dentro del microcontrolador, se ha incluido en su arquitectura para

Prioridad	Símbolo	Dirección	Vector
Más baja	IF15	$FFDE	Vector alto de fin de conversión del ADC
		$FFDF	Vector bajo de fin de conversión del ADC
	IF14	$FFE0	Vector alto del teclado
		$FFE1	Vector bajo del teclado
	IF13 a IF6	-	Espacio sin uso
	IF5	$FFF2	Vector alto de sobreflujo del TIM
		$FFF3	Vector bajo de sobreflujo del TIM
	IF4	$FFF4	Vector alto del canal 1 del TIM
		$FFF5	Vector bajo del canal 1 del TIM
	IF3	$FFF6	Vector alto del canal 0 del TIM
		$FFF7	Vector bajo del canal 0 del TIM
	IF2	-	Espacio sin uso
	IF1	$FFFA	Vector alto de la $\overline{IRQ}$
		$FFFB	Vector bajo de la $\overline{IRQ}$
	-	$FFFC	Vector alto de SWI
		$FFFD	Vector bajo de SWI
	-	$FFFE	Vector alto del *reset*
Más alta		$FFFF	Vector abajo del *reset*

Tabla 20.1 *Tabla de vectores de las interrupciones en los microcontroladores 68H(R)C08JK1/JK3/JL3*

que el usuario pueda probar la ejecución correcta de un programa por medio de una interfaz de comunicación serial de dos hilos RS232 y el puerto de una computadora como dispositivo maestro. Las principales características del modo monitor son:

1. Mantiene la funcionabilidad de todos los *pines*, reserva un *pin* (PTB0) para establecer la comunicación serial entre el monitor y la PC.
2. Usa un protocolo de comunicación estándar marca-espacio sin retorno a cero (NRZ).
3. Efectúa la comunicación a una velocidad programable entre 4.800 baudios y 28,8 Kbaudios, y puede ejecutar códigos de programa desde las memorias RAM o ROM.

En la **figura 20.3** se muestra el circuito típico que se requiere para trabajar con el microcontrolador en el modo monitor, en él se han usado un *driver* de comunicación RS232 (MC145407) y un *buffer* de tres estados (74125) para controlar la dirección de los datos.

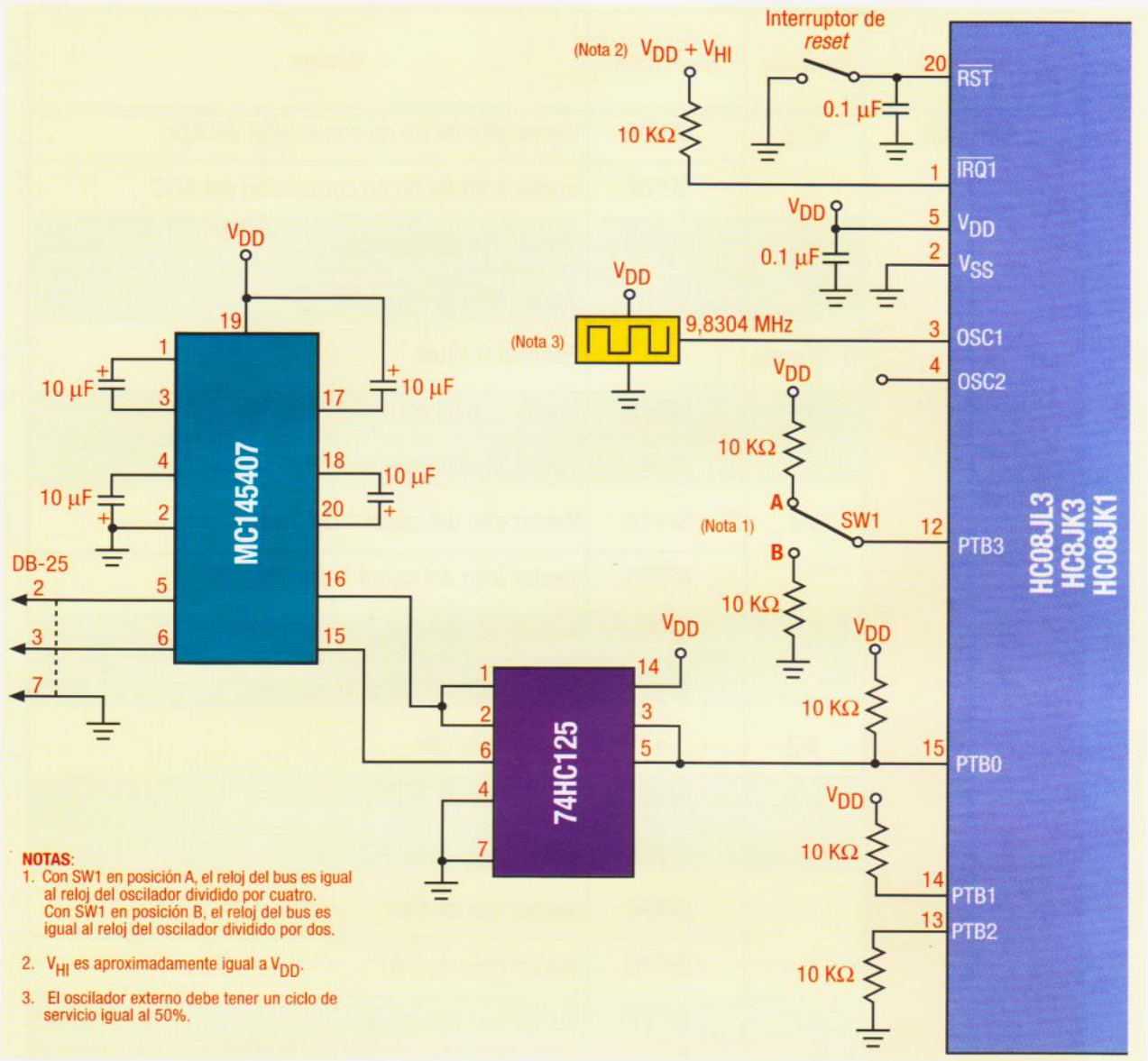


Figura 20.3 *Circuito típico para establecer el uso del modo monitor*

Acceso al modo monitor

La **tabla 20.2** resume las condiciones que se deben cumplir para entrar al modo monitor. Para entrar a dicho modo de trabajo, de acuerdo con la figura 20.3, el microcontrolador requiere la aplicación de un *reset* a la conexión (*power-on reset* POR), luego de haber cumplido con uno de los dos conjuntos de condiciones que a continuación se resumen.

1. Colocar la principal entrada externa de interrupción $\overline{IRQ1}$, localizada en el pin 1 del microcontrolador, en un nivel de voltaje igual a V_{DD} +V_{HI} (V_{HI} puede estar entre 0,7xV_{DD} como valor mínimo, y V_{DD} como máximo), excitar la entrada de reloj externa a través de OSC1 con una frecuencia igual a 4,9125 MHz, y llevar el pin de entrada PTB3 a un nivel bajo (0 lógico).
2. Colocar la entrada externa de interrupción $\overline{IRQ1}$ en un nivel de voltaje igual a V_{DD} +V_{HI}, excitar la entrada de reloj externo a través de OSC1 con un generador de frecuencia externo ajustado en 9,8304 MHz, y llevar el pin de entrada PTB3 a un nivel alto (1 lógico).

$\overline{IRQ1}$	PTB3	PTB2	PTB1	PTB0	Fuente del reloj y frecuencia	Frecuencia del bus	Comentarios
$V_{DD} + V_{HI}$	0	0	1	1	OSC1 a 4,9152 MHz	2,4576 MHz	Entra al modo monitor: El reloj externo llega directamente a OSC1. La comunicación se establece a 9,600 baudios. El temporizador COP permanece deshabilitado.
$V_{DD} + V_{HI}$	1	0	1	1	OSC1 a 9,8304 MHz	2,4576 MHz	
V_{DD}	X	X	X	X	Frecuencia definida en los módulos osciladores XTAL ó RC	XTALCLK ÷ 4 ó RCCLK ÷ 4	Entra al modo usuario

Tabla 20.2 *Requerimientos y opciones para entrar al modo monitor*

Vectorización en el modo monitor

El MCU cambia algunas direcciones definidas en el modo usuario para los vectores de las interrupciones del *reset*, el SWI, y el *break*; éstos son definidos alternativamente en la página $FE, y permiten que la CPU inicialice la ejecución del código del programa desde el *firmware* del monitor, y no desde la memoria de programa del modo usuario.

Velocidad de comunicación del modo monitor

La velocidad con la que se efectúa la transferencia de datos entre el microcontrolador y la PC, se define en función de la frecuencia del módulo oscilador activo, y depende también del estado de la entrada PTB3, una vez se ha entrado al modo monitor. Cuando el PTB3 está en estado alto se divide por 1.024, mientras que cuando su nivel es un cero lógico se divide por 512; el resumen para la selección de la rata de transmisión del modo monitor, de acuerdo con el circuito de la figura 20.3.

Formatos básicos de comunicación del modo monitor

En la **figura 20.4a**, se puede apreciar que la comunicación con el programa monitor se efectúa en un formato de datos estándar de marca-espacio sin retorno de cero (NZR). La velocidad de transmisión puede ser cualquier valor estándar entre 4.800 baudios y 28,8 Kbaudios.

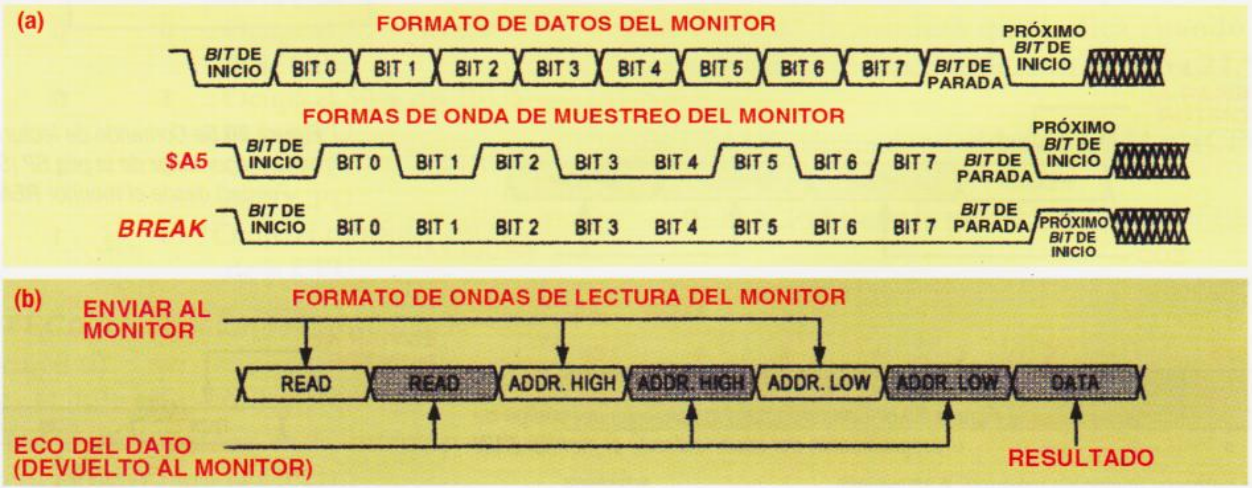


Figura 20.4 *Formato de datos y eco durante el ciclo de lectura desde el modo monitor*

Para garantizar la transferencia limpia de datos por medio del *driver* de comunicación, el programa monitor hace eco de cualquier dato recibido, enviándolo de vuelta hacia el terminal PTB0 para efectuar el chequeo de error respectivo. Esto se puede apreciar en la **figura 20.4b**, allí se ve que cualquier resultado de un comando sólo será transferido luego del eco del último *byte* del mismo.

Comandos del monitor

En las **figuras 20.5** se describen gráficamente las secuencias de los seis comandos que se usan en el modo monitor: lectura de memoria **READ**, escritura en memoria **WRITE**, lectura indexada **IREAD**, escritura indexada ***IWRITE***, lectura del puntero de la pila **READSP**, y ejecución del programa **RUN**.

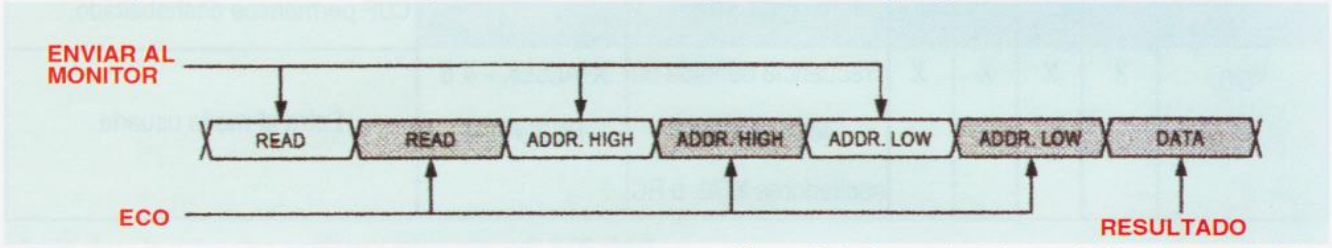


Figura 20.5a *Comando de lectura del área de memoria* ***READ***

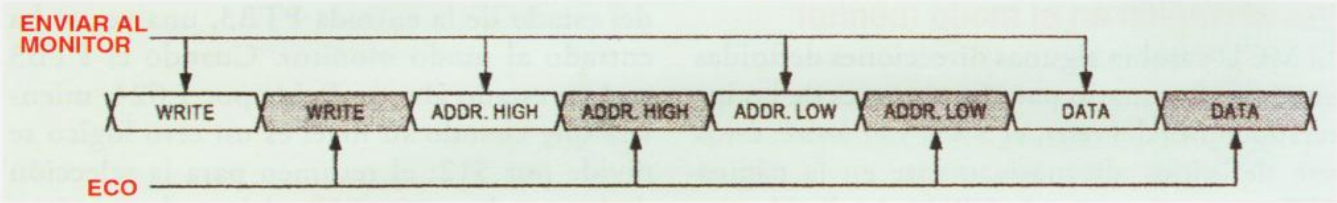


Figura 20.5b *Comando de escritura del área de memoria* ***WRITE***

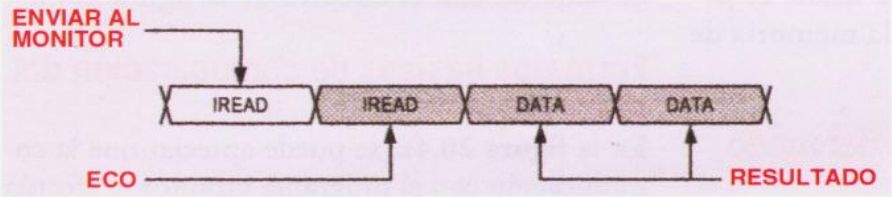


Figura 20.5c *Comando de lectura indexada del área de memoria* ***IREAD***

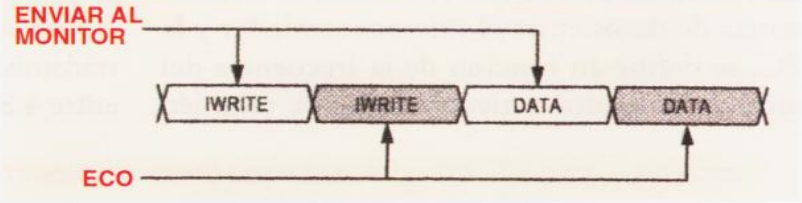


Figura 20.5d *Comando de escritura indexada del área de memoria* ***IWRITE***

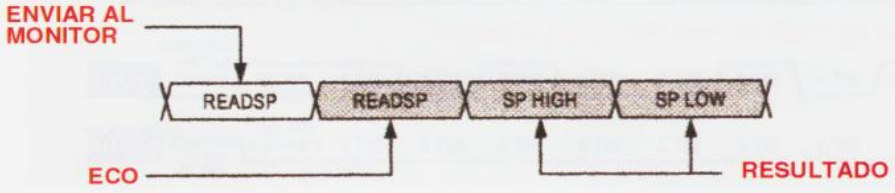


Figura 20.5e *Comando de lectura del registro apuntador de la pila SP (Stack Pointer) desde el monitor* ***READSP***

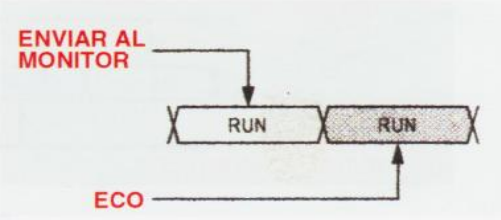
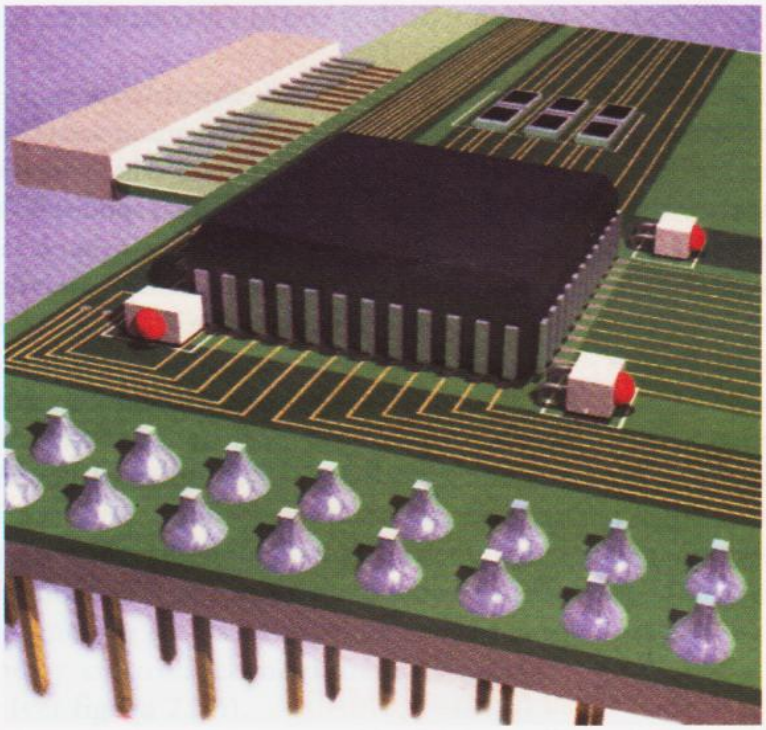


Figura 20.5f *Comando para la ejecucón del código de programación del usuario desde el monitor* ***RUN***

Figura 20.5 *Formato de datos y eco durante el ciclo de lectura desde el modo monitor*

LECCIÓN 21



EL MÓDULO DMA Y LOS MODOS DE DIRECCIONAMIENTO EN LOS MICROCONTROLADORES 68HC08

Los microcontroladores Motorola de la familia 68HC08, además de poseer dieciseis modos de direccionamiento, tienen un módulo especial para el acceso directo a memoria (DMA); que, tal como en los grandes sistemas abiertos de cómputo, se usa para transferir rápidamente bloques de datos entre áreas y registros de memoria.

La característica más importante con que cuentan los microcontroladores, desde el punto de vista del programador y en cuanto al manejo y al procesamiento de los datos, se encuentra en la disponibilidad de varios modos de direccionamiento e instrucciones que soportan la transferencia de datos entre los registros y la memoria.

Acceso directo a memoria (DMA)

El acceso directo a memoria DMA (*Direct Memory Access*), es un método rápido y eficaz que permite guardar y retirar grandes volúmenes de datos de la memoria y algunos registros de datos especiales, sin la intervención de la CPU. El DMA es un método que nace con los sistemas de cómputo grandes y complejos y los microprocesadores de alto rango; afortunadamente el desarrollo de la tecnología y las exigencias de los sistemas embebidos de control en un solo *chip* (circuito integrado) cada vez mayores y más frecuentes, han impulsado la integración dentro de la estructura de los MCU (*Microcontroller Unit*) de funciones y módulos periféricos de alto desempeño, tal como el DMA.

Los microcontroladores de 8 *bits,* de la familia 68HC08, son un ejemplo de la evolución en el manejo de la memoria; éstos poseen un módulo que permite el coprocesamiento de datos por medio del acceso directo, que se conoce como el módulo DMA08.

En la **figura 21.1** se muestra un diagrama de bloques simplificado de un sistema general con DMA; en este caso es claro que la arquitectura de los buses de datos, direcciones y control, son un soporte fundamental para los movimientos de datos que se apoyan en el DMA. Para que la transferencia con el DMA se efectúe, este módulo debe tomar el control maestro de todo el sistema de buses y asumir temporalmente las funciones de control de la CPU; en términos generales, el DMA opera de dos maneras diferentes: en ciclo compartido y con la CPU detenida.

Para transferir pequeños volúmenes de datos el método de acceso directo a memoria que más se usa es con la CPU detenida; ella le otorga temporalmente al DMA el control total del sistema de buses y le permite hacer la transferencia en el menor tiempo posible; mientras que cuando se va a transferir un gran bloque de información, el método que se usa sólo toma el control de una parte del sistema de buses y lo comparte con la CPU, permitiendo así que desde ésta se desarrollen las tareas y el procesamiento continuo de los códigos de programa sin interrupción durante la transferencia. Bajo esta condición la velocidad del movimiento de datos se reduce, pero la CPU no detiene la ejecución de sus tareas.

El módulo DMA08

El uso del DMA es importante durante el movimiento de datos, porque libera de funciones a la CPU, reduce el código ejecutable necesario, incrementa la velocidad de respuesta, y permite usar con mejor eficiencia el espacio de memoria de programa disponible. En los MCU de la familia

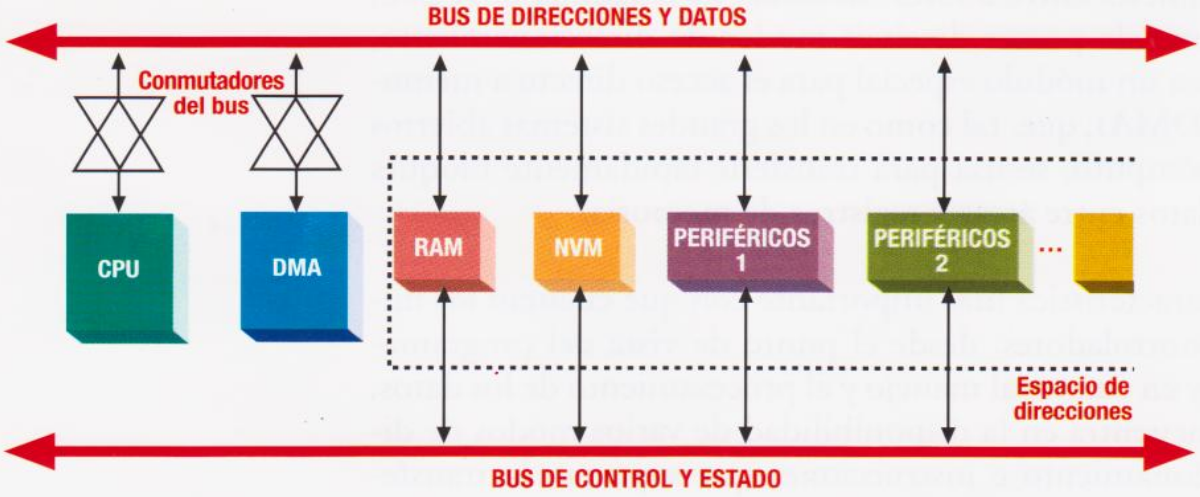


Figura 21.1 *Diagrama de bloques de un sistema simple con DMA*

68HC08, el DMA08 es un módulo coprocesador que puede tomar el control de los buses de dirección y datos al concluir cualquier ciclo de bus, y está diseñado para permitir la transferencia de datos entre dos posiciones cualesquiera que puedan ser direccionadas por la CPU.

El acceso al módulo DMA08 está basado en una característica modular de la estructura interna del MCU de la familia 68HC08, que permite desde fábrica, habilitar y expandir el número de canales de acceso directo a memoria DMA, entre uno y siete canales independientes.

La **figura 21.2** es una representación simplificada del módulo DMA08 que muestra sus bloques funcionales desde el punto de vista del programador; en él, el sistema de control lógico habilita el DMA y selecciona el conjunto de registros que se requieren por el canal activo, maneja el incremento o decremento del registro base apuntador de direcciones de la fuente y el destino, y controla el movimiento seguro de datos (ver **figura 21.3**).

Operación del módulo DMA08

En la figura 21.3 se representa claramente en bloques el modo de operación del DMA; los bloques que representan **las direcciones bases de las posiciones fuente y destino**, están conformados por dos registros de 16 *bits* cada uno, en donde la CPU coloca la dirección base antes de iniciar la transferencia con el DMA; mientras que el **bloque contador de *bytes*** está constituido por un registro que lleva la cuenta del número de transferencias hechas. En condiciones normales, la ALU del DMA aumentará en uno el contenido del contador cada vez que un *byte* complete su transferencia, y tras el ciclo de *reset* se borrará.

El bloque de longitud define, por medio del registro de longitud de bloque, el número de *bytes* que van a ser transferidos en una operación particular del DMA, y su contenido se usa para ser comparado con el del contador de *bytes* y así determinar el fin de la transferencia. En el DMA, la ALU es un subsistema de 16 *bits* que se usa para calcular rápidamente las

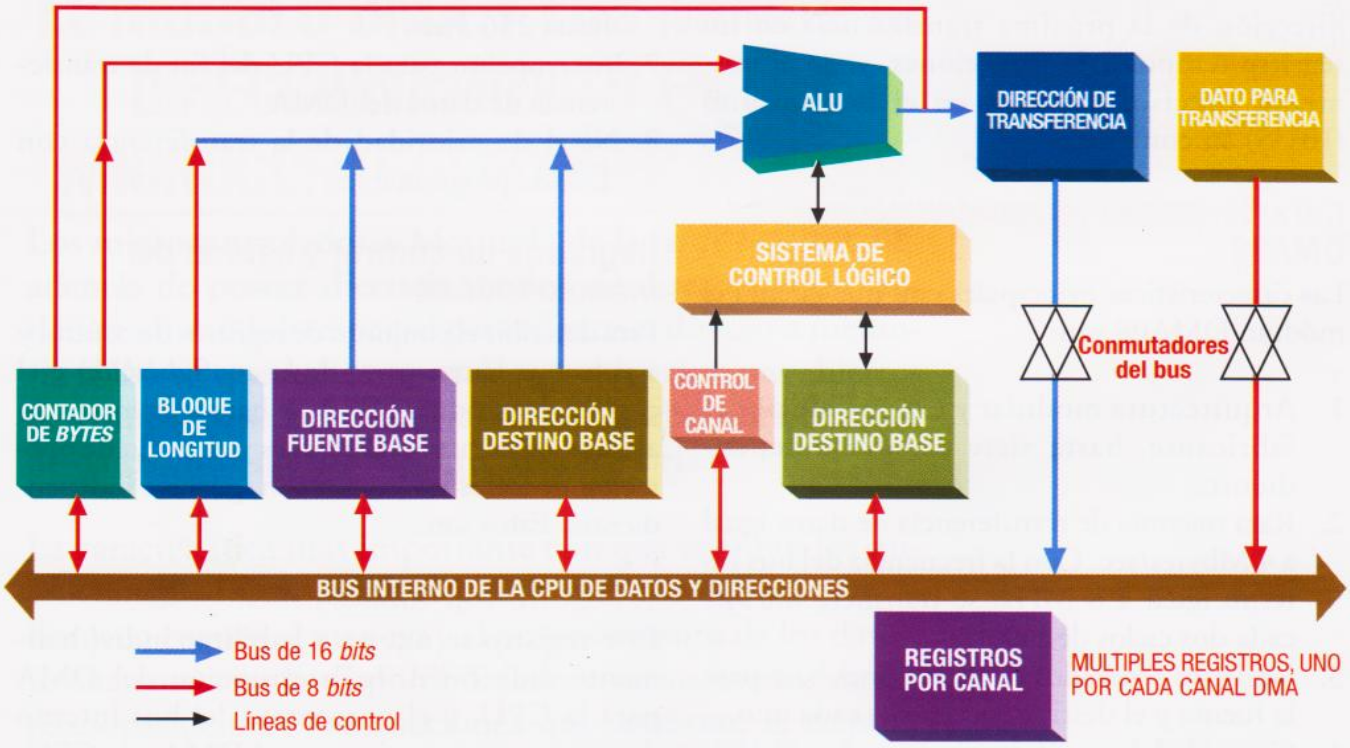


Figura 21.2 *Diagrama en bloques del módulo DMA08*

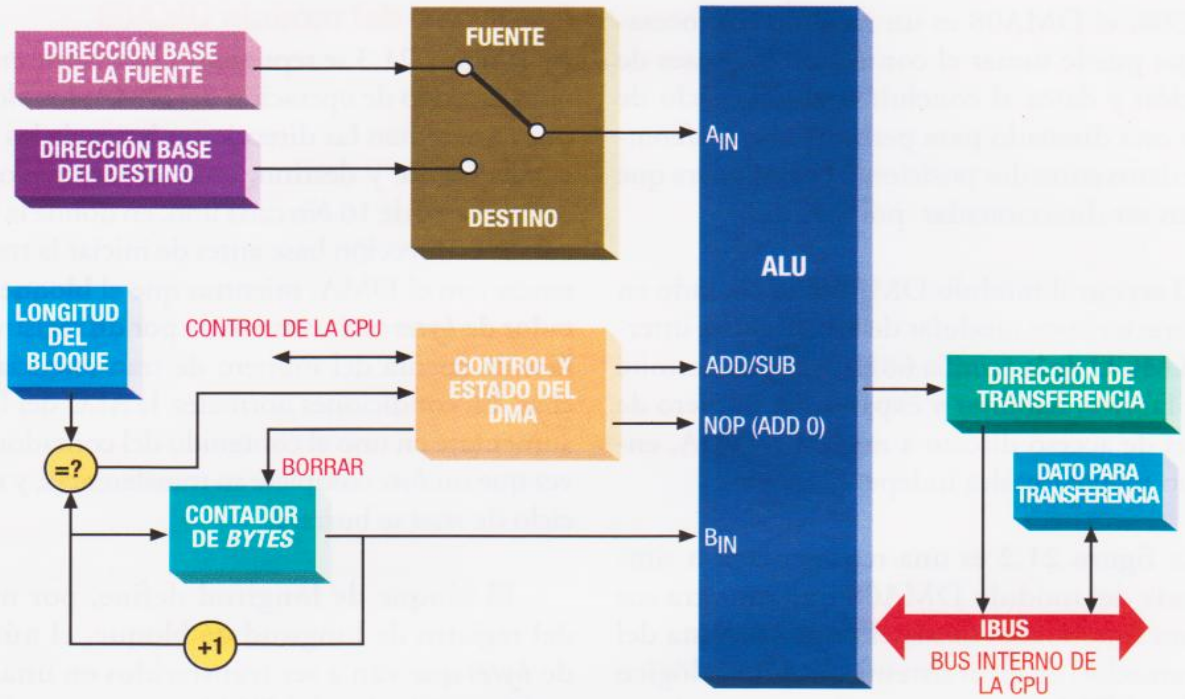


Figura 21.3 *Operación del módulo DMA08*

direcciones actuales de la fuente y el destino (en un sólo ciclo), y para incrementar el contador de *bytes* con cada transferencia. Bajo la dirección del módulo DMA, la ALU guarda la dirección de la próxima transferencia en un registro temporal de direcciones, y, en el momento preciso, deposita en el bus interno (IBUS) su contenido.

Características resumidas del módulo DMA08

Las características principales con que cuenta el módulo DMA08 son:

1. Arquitectura modular y expansible por el fabricante, hasta siete canales independientes.
2. Rata máxima de transferencia de datos igual a 4 Mbytes/sec. Con la frecuencia del bus interno igual a 8 MHz, se transfiere un *byte* cada dos ciclos de bus.
3. Registros separados de direcciones base para la fuente y el destino, de 16 *bits* cada uno.
4. Capacidad de transferencia de un *byte* (8 *bits*), o una palabra (16 *bits*).
5. Opción de transferencia de un sólo bloque, o de bloques repetidos en lazo de seguridad.
6. Longitud del bloque de datos programable hasta 256 *bytes*.
7. Interrupción para la CPU del fin de transferencia de datos del DMA.
8. Nivel de prioridad de la transferencia con DMA, programable.

Registros de control y estado del módulo DMA08

Para describir el conjunto de registros de control y estado que hacen parte de la configuración y el control del módulo DMA, se hará referencia sólo al canal 0, ya que las funciones y el modo de operación de todos los canales son iguales e independientes. Estos son:

1. Registro 1 de control del DMA (DC1)

Este registro se usa para habilitar individualmente cada canal, la interrupción del DMA para la CPU, y el porcentaje del bus interno que se va a compartir entre el DMA y la CPU. (**Figura 21.4**)

PARA SABER MÁS:

Algoritmo básico para la transferencia con el módulo DMA08

Para programar la transferencia de datos, a través del módulo DMA08, se puede usar el siguiente procedimiento como una guía, y en vista de que todos los canales del DMA operan idénticamente, éste se describirá sólo para el canal 0.

Paso 1: Inhabilite el canal 0 del DMA.

Paso 2: Escriba en el registro de dirección base de la fuente, la dirección del primer dato que debe ser leído desde la fuente (**D0SH** y **D0SL**).

Paso 3: Escriba en el registro de dirección base del destino, la dirección de la posición en donde se debe escribir el primer dato destino (**D0DH** y **D0DL**).

Paso 4: Acceda el registro de control del canal 0 (**D0C**) para habilitarlo, y seleccionar las fuentes del incremento y decremento, y la longitud del dato.

Paso 5: Defina en el registro de longitud del bloque del canal 0 (**D0BL**), el número de *bytes* por transferir.

Paso 6: Use el registro de estado y control del DMA (**DSC**) para asignar el nivel de prioridad de la transferencia, activar la transferencia cíclica, y habilitar o no el DMA durante el modo de espera del MCU (WAIT).

Paso 7: Para controlar desde el *software* la transferencia de datos, coloque en 1 lógico el *bit* **SWI0** del registro 2 de control del DMA (**DC2**). La transferencia comenzará dos ciclos después de la habilitación del canal 0.

Paso 8: Finalmente, para habilitar la transferencia y manejarla desde un periférico por medio de una interrupción, se debe habilitar la interrupción de fin de transferencia, seleccionar el porcentaje de ocupación del bus y habilitar el canal DMA. Todo esto, desde el registro1 de control del DMA (**DC1**).

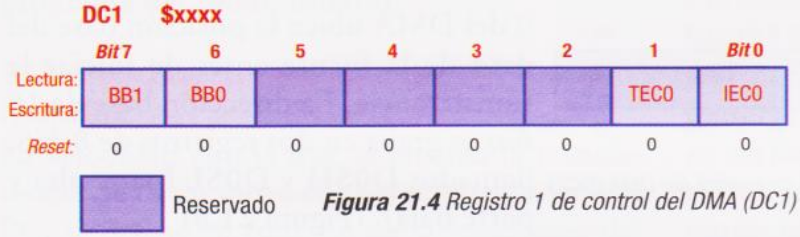


Figura 21.4 Registro 1 de control del DMA (DC1)

Bits **BB1** **BB0** (selectores del ancho de banda para el DMA)

BB1	BB0	
0	0	: Ocupa el 25% del IBUS de la CPU
0	1	: Ocupa el 50% del IBUS de la CPU
1	0	: Ocupa el 67% del IBUS de la CPU
1	1	: Ocupa el 100% del IBUS de la CPU

Bit **TEC0** (habilitador de transferencia del canal 0)

1: Habilitado, y **0**: Inhabilitado

Bit **IEC0** (habilitador de la interrupción del DMA para la CPU)

1: Habilita la interrupción del canal 0, y

0: Inhabilita la interrupción del canal 0

2. Registro de estado y control del DMA (DSC)

Con este registro, se programan: la prioridad, el modo de transferencia, y la operación del DMA, cuando la CPU está en espera (o en modo WAIT); y en uno de sus *bits* está la bandera que indica cuando la transferencia se ha completado. (**Figura 21.5**).

Bit **DMAP** (selector de prioridad del DMA y la CPU)

1: El DMA tiene la prioridad, y

0: La CPU tiene la prioridad

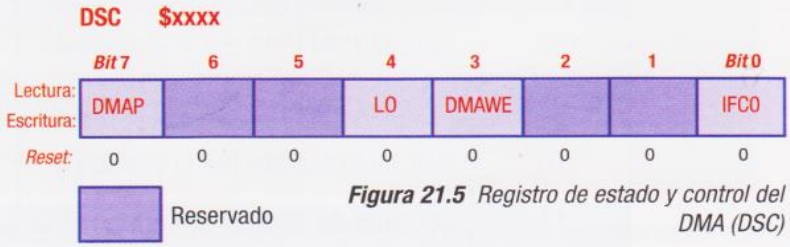


Figura 21.5 Registro de estado y control del DMA (DSC)

Bit **L0** (*bit* habilitador del modo de transferencia cíclica del canal 0)

1: Habilita el modo cíclico, y

0: Habilita el modo bloque (inhabilita el modo cíclico)

Bit **DMAWE** (habilitador del DMA durante el modo WAIT)

1: Transferencia habilitada, y

0: Transferencia suspendida

Bit **IFC0** (bandera de interrupción del canal 0)

1: Transferencia completa, y

0: Transferencia incompleta

3. Registro 2 de control del DMA (DC2)

Este registro se usa para configurar las fuentes y el comienzo de la transferencia (**Figura 21.6**)

bits **SWI7** a **SWI0** (habilitadores por *software* del inicio de transferencia del DMA)

1: Inicia la transferencia, y

0: Bloquea la transferencia

DC2 $xxxx

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	SWI7	SWI6	SWI5	SWI4	SWI3	SWI2	SWI1	SWI0
Reset:	0	0	0	0	0	0	0	0

Figura 21.6 *Registro 2 de control del DMA (DC2)*

4. Registro de control del canal 0 del DMA (D0C)

Por medio de los cuatro *bits* **SDC3** a **SDC0** de este registro, se calculan las direcciones base de la fuente y del destino para el canal 0. Por medio de él, también se configura un *byte* o una palabra para la transferencia de datos, se definen las fuentes para la solicitud del DMA, y se controla el desplazamiento del registro de direcciones temporal que el DMA usa como apuntador del bloque. (**Figura 21.7** y **Tabla 21.1** y **21.2**)

Bit **BWC** (*bit* de control de la longitud del dato transferido)

1: El DMA transfiere datos de 16 *bits*, y

0: El DMA transfiere datos de 8 *bits*

D0C $xxxx

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	SDC3	SDC2	SDC1	SDC0	BWC	DTS2	DTS1	DTS0
Reset:	Estado indefinido después del *reset*							

Figura 21.7 *Registro de control del canal 0 (D0C)*

5. Registros fuente del DMA con la dirección base, del canal 0 (D0SH y D0SL)

Por medio de estos registros, el canal 0 del DMA ubica la posición base del dato de la fuente antes de iniciar la transferencia. La dirección base de 16 *bits* se graba en dos registros de 8 *bits*, llamados **D0SH** y **D0SL** (parte alta y parte baja). (**Figura 21.8**)

DTS[2:0]	Fuentes de solicitud de transferencia del DMA	Inicializadores por *software*
000	Solicitud del DMA por la entrada 0	SWI0
001	Solicitud del DMA por la entrada 1	SWI1
010	Solicitud del DMA por la entrada 2	SWI2
011	Solicitud del DMA por la entrada 3	SWI3
100	Solicitud del DMA por la entrada 4	SWI4
101	Solicitud del DMA por la entrada 5	SWI5
110	Solicitud del DMA por la entrada 6	SWI6
111	Solicitud del DMA por la entrada 7	SWI7

Tabla 21.1 *Selección de fuentes para la solicitud de la transferencia desde el DMA*

SDC[3:0]	Dirección fuente	Dirección destino
1010	Incremento	Incremento
1001	Incremento	Decremento
1000	Incremento	Estático
0110	Decremento	Incremento
0101	Decremento	Decremento
0100	Decremento	Estático
0010	Estático	Incremento
0001	Estático	Decremento
0000	Estático	Estático

Tabla 21.2 *Bits de control de direcciones base del DMA*

D0SH $xxxx

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8

Reset: Se borra el registro después del *reset*

D0SL $xxxx + 1

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

Reset: Se borra el registro después del *reset*

Figura 21.8 *Registros de dirección base fuente (D0SH y D0SL)*

6. Registros destino del DMA con la dirección base, del canal 0 (D0DH y D0DL)

Teniendo en cuenta la dirección de 16 *bits* contenida en los registros **D0DH** y **D0DL** (parte alta y parte baja respectivamente), el canal 0 escribe en la posición destino base el primer dato transferido por el DMA.

D0DH $xxxx

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8

Reset: Se borra el registro después del *reset*

D0DL $xxxx + 1

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

Reset: Se borra el registro después del *reset*

Figura 21.9 *Registros de dirección base destino (D0DH y D0DL)*

7. Registro de control de longitud del bloque, del canal 0 (D0BL)

El número de *bytes* que serán transferidos en un bloque por el canal 0 del DMA, se controlan desde este registro. Al escribir el valor $00 en el registro **D0BL** se define la mayor longitud para el bloque, que es de 256 *bytes*.

D0BL $xxxx

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	BL7	BL6	BL5	BL4	BL3	BL2	BL1	BL0

Reset: Se borra el registro después del *reset*

Figura 21.10 *Registro de longitud del bloque del DMA (D0BL)*

8. Registro contador de bytes transferidos, del canal 0 (D0BC)

Durante el proceso de transferencia del DMA, en el contenido del registro contador de *8 bits* **D0BC** se lleva la cuenta del número de datos de un *byte* que se han transferido a través del canal 0.

Este registro contador se puede borrar por tres causas, que son: por un ciclo de *reset* interno o externo, cuando alcanza la longitud definida para el bloque de datos, y cuando se escribe una nueva dirección base fuente o destino.

D0BC $xxxx

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: Escritura:	BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0

Reset: Se borra el registro después del *reset*

Figura 21.11 *Registro contador de bytes transferidos del DMA (D0BC)*

Los modos de direccionamiento en la familia HC08

En los microcontroladores 68HC08 de Motorola, hay disponibles dieciséis modos de direccionamiento que le otorgan una gran flexibilidad y eficiencia al manejo de datos que el programador puede efectuar con estos dispositivos. En términos de la CPU, los modos de direccionamiento del MCU son las opciones con que ella cuenta para localizar los datos que se requieren para ejecutar cualquiera de sus instrucciones.

Los dieciséis modos de direccionamiento son:

- **Inherente**
- **Inmediato**
- **Directo**
- **Extendido**
- **Indexado sin desplazamiento u *offset***
- **Indexado con *offset* de 8 *bits***
- **Indexado con offset de 16 *bits***
- **Del SP con *offset* de 8 *bits***
- **Del SP con *offset* de 16 *bits***
- **Relativo**
- **Memoria a memoria, inmediato a directo**
- **Memoria a memoria, directo a directo**
- **Memoria a memoria, indexado a directo con incremento**
- **Memoria a memoria, directo a indexado con incremento**
- **Indexado con incremento posterior**
- **Indexado con *offset* de 8 *bits* e incremento posterior**

Direccionamiento inherente

En los MCU HC08 hay 54 instrucciones inherentes. Todas las instrucciones que soportan este modo de direccionamiento se denominan instrucciones inherentes y se caracterizan porque no requieren un ciclo de búsqueda (*fetch*) para el operando, trabajan sobre registros internos de la CPU, tales como el acumulador A, el registro índice H:X, el apuntador de la pila SP, y el registro del código de condiciones CCR. La principal ventaja de las instrucciones inherentes, es que no requieren de un direccionamiento a memoria y, por lo tanto, su tiempo de captura y ejecución es el menor.

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
A657	EX_1	LDA	#$57	;A = $57
AB45		ADD	#$45	;A = $9C
72		DAA		;A = $02 w/acarreo ;coloca el bit en $102
A614	EX_2	LDA	#20	;LS dividendo de A
8C		CLRH		;Borra el MS del dividendo
AE03		LDX	#3	;Divisor de X
52		DIV		;(H:A)/X→A=06,H=02
A630	EX_3	LDA	#$30	;A = $30
87		PSHA		;Guarda el dato $30 en ;la pila y decrementa ;el apuntador SP en 1

Figura 21.12 *Ejemplos de instrucciones del modo de direccionamiento inherente*

En la **figura 21.12** se muestran tres pequeños segmentos de programas en lenguaje ensamblador que son ejemplos del modo de direccionamiento inherente; en ésta se destacan con color oscuro las líneas de programa que contienen instrucciones inherentes. Las instrucciones son: **DAA** (ajuste decimal al registro acumulador), **CLRH** (borrar el *byte* alto H del registro índice), y **PSHA** (guarda el contenido del acumulador en la pila).

Nota:
En éste y en todos los demás programas escritos en lenguaje ensamblador, se usará el símbolo (#) antes de cualquier número, para indicar que es un operando inmediato; y, de acuerdo con la sintaxis definida por los ensambladores de Motorola, también se usará el símbolo ($) antes de cada número para representarlo en notación hexadecimal.

Direccionamiento inmediato

Las instrucciones de direccionamiento inmediato en los MCU HC08 son 18. Son todas aquellas que contienen el operando en los *bytes* que van inmediatamente después del código de la operación; los datos inmediatos pueden ser de uno o dos *bytes* de acuerdo con el tamaño del registro involucrado en la instrucción, y pueden representar un valor o apuntar a la dirección de un valor.

El programa ejemplo que se muestra en la **figura 21.13** contiene dos instrucciones inmediatas: **AIX** (suma un valor inmediato con signo al registro índice), y **CPHX** (compara el registro índice con un valor inmediato); en éste, primero se borra el registro índice (H:X) y luego se

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
5F	START	CLRX		;X = 0
8C		CLRH		;H = 0
AF01	TAG	AIX	#1	;(H:X) = (H:X) + 1
65FFFF		CPHX	#$FFFF	;Compara (H:X) con ;$FFFF
26F9		BNE	TAG	;Ejecución cíclica ;hasta su igualdad
20F5		BRA	START	;Salto a inicio

Figura 21.13 *Ejemplos de instrucciones del modo de direccionamiento inmediato*

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
	RAM	EQU	$50	;Asignación a la RAM
	ROM	EQU	$6E00	;Asignación a la ROM
		ORG	$RAM	;Comienzo de la RAM
	TEMP	RMB	2	;Reserva 2 bytes
		ORG	$ROM	;Comienzo de la ROM
5F	START	CLRX		;X = 0
8C		CLRH		;H = 0
3550		STHX	TEMP	;H:X=0 > TEMP
455555		LDHX	#$5555	;Carga H:X con $5555
3550		STHX	TEMP	;TEMP=$5555
7550	BAD_PART	CPHX	RAM	;RAM=TEMP
26FC		BNE	BAD_PART	;RAM=TEMP a menos que ;ocurra algún error
20F1		BRA	START	;Inicia de nuevo el ciclo

Figura 21.14 *Ejemplos de instrucciones del modo de direccionamiento directo*

aumenta en uno hasta alcanzar el valor $FFFF. Cuando la condición de comparación definida por la instrucción CPHX es verdadera, el programa salta a la línea de inicio (START) y repite el proceso indefinidamente.

Direccionamiento directo

Los MCU HC08 de Motorola poseen 37 instrucciones directas. En este modo de direccionamiento el primer *byte* es el código de la operación, y el segundo es el *byte* bajo de la dirección efectiva del operando; ya que el *byte* más alto de la dirección se asume como $00 y no se incluye en la instrucción como un *byte* más, el tiempo de ejecución es sólo de un ciclo.

Por lo anterior, el uso del direccionamiento directo queda limitado a la **página cero** o **directa** del área de memoria que comprende las primeras 256 localidades (entre las posiciones $0000 y $00FF).

En el programa ejemplo, de la **figura 21.14**, se pueden reconocer dos instrucciones propias del modo de direccionamiento directo, que son: **STHX** (guardar el registro índice en la memoria), y CPHX (compara el registro índice con un valor en memoria).

La primera instrucción STHX localiza la etiqueta TEMP en la posición cero de la RAM, y la segunda carga en dicha posición el dato $5555; mientras que la instrucción CPHX compara el valor contenido en H:X con el de la posición RAM +1 (en este caso, RAM +1=TEMP=posiciones $50 y $51=$5555).

Direccionamiento extendido

El programador de los MCU HC08 dispone de 16 instrucciones que son la base del direccionamiento extendido.

Las instrucciones llamadas extendidas, pueden acceder a cualquier dirección del mapa de memoria de 64 Kbytes con que cuentan estos microcontroladores; todas las instrucciones son de 3 *bytes* de longitud y en ellas el primer *byte* es el código de operación; mientras que los dos siguientes son los *bytes* más y menos significativos de la dirección efectiva del operando.

En todos los ensambladores eficientes, el programador puede olvidarse de definir si la instrucción es directa o extendida, ya que el programa ensamblador automáticamente decidirá la forma más corta de la instrucción de acuerdo con el límite que está definido para la página cero ($00FF).

En la **figura 21.15** se muestran dos ejemplos de este modo de direccionamiento usando la instrucción **LDX** (carga el *byte* bajo X del registro índice con el contenido de una localidad de memoria).

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
		ORG	$50	;Inicia en $50
		FCB	$FF	;$50 = $FF
5F		CLRX		
BE50		LDX	$0050	;Carga directa ;del registro X ;con el dato $0050
		ORG	$6E00	;Inicio en $6E00
		FCB	$FF	;$6E00 = $FF
5F		CLRX		
CE6E00		LDX	$6E00	;Carga extendida ;del registro X

Figura 21.15 *Ejemplos de instrucciones del modo de direccionamiento extendido*

Direccionamiento indexado, sin desplazamiento

En la familia HC08 hay 29 instrucciones para el direccionamiento indexado sin desplazamiento u *offset*. Todas las instrucciones que soportan este modo son de un *byte* y se usan para acceder a datos con dirección variable, tal como en el caso de las tablas. Las instrucciones indexadas sin *offset* se usan generalmente para mover apuntadores a través de una tabla de datos localizada en memoria.

En esta familia se han dispuesto dos registros (H y X), de un *byte* cada uno, que contienen las direcciones condicionales alta y baja del operando, y, gracias al registro H, este modo de direccionamiento no queda limitado a las primeras 256 posiciones de memoria, tal como sucede en la familia HC05.

Para mantener la compatibilidad absoluta entre las instrucciones de los MCU de Motorola de las familias HC05 y HC08, el registro H se mantiene en cero cuando no se usa ninguna instrucción que lo afecte por parte del HC08.

En la **figura 21.16** se muestra como ejemplo una línea de código de programa que ilustra el modo de direccionamiento indexado sin *offset*, por medio de la instrucción de salto incondicional JMP.

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
FC		JMP	,X	;SIN OFFSET ;salta a la ;dirección ;apuntada por el ;registro H:X

Figura 21.16 *Ejemplo del modo de direccionamiento indexado sin offset*

Direccionamiento indexado, con desplazamiento de 8 *bits*

Las instrucciones para el direccionamiento indexado con *offset* de 8 *bits* son 29. Todas las instrucciones de este modo de direccionamiento son de dos *bytes*, y al igual que las que no usan desplazamiento; éstas también se usan para acceder a datos con dirección variable.

Para determinar la dirección efectiva del operando, la CPU usa el resultado que se obtiene al sumar los *bytes* sin signo del registro índice (H:X), con el *byte* sin signo que precede al código de operación de la instrucción.

Estas instrucciones se usan a menudo para seleccionar el dato o elemento n-esimo dentro de una tabla de k elementos, en este caso el n-esimo valor deberá estar típicamente en el registro H:X y la dirección de inicio de la tabla deberá estar en el *byte* siguiente al código de operación de la instrucción.

Por lo anterior, este direccionamiento quedará limitado a las primeras 256 posiciones de memoria, pero, sin embargo, la tabla puede localizarse en cualquier lugar del mapa de memoria al usar el registro índice como dirección base, y el siguiente *byte* como desplazamiento.

En la **figura 21.17** se muestra como ejemplo una línea de código de programa que ilustra el modo de direccionamiento indexado con *offset* de 8 *bits*, por medio de la instrucción de salto incondicional JMP.

Direccionamiento indexado, con desplazamiento de 16 *bits*

El MCU HC08 tiene 16 instrucciones de direccionamiento indexado con *offset* de 16 *bits*. En este modo de direccionamiento todas las instrucciones son de tres *bytes* y pueden acceder a cualquier dato con dirección variable en toda el área de memoria; la CPU suma el contenido de H:X con la pa-

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
ECFF		JMP	$FF,X	;OFFSET DE 8-BITS ;Salta la dirección ;apuntada por el ;registro H:X + $FF

Figura 21.17 *Ejemplo del modo de direccionamiento indexado con offset de 8 bits*

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
DC10FF		JMP	$10FF,X	;OFFSET DE 16-BITS ;salta a la dirección ;apuntada por el ;registro H:X + $10FF

Figura 21.18 *Ejemplo del modo de direccionamiento indexado con offset de 16 bits*

labra formada por los dos *bytes* que siguen al código de operación para encontrar la dirección efectiva del operando.

En este direccionamiento indexado con *offset* de 16 *bits,* las instrucciones se usan de la misma forma que en el direccionamiento anterior, para localizar y capturar datos dentro de una tabla, pero, en este caso, con movilidad en todos los 64 Kbytes de memoria.

En la **figura 21.18** se muestra como ejemplo una línea de código de programa que ilustra el modo de direccionamiento indexado con *offset* de 16 *bits*, usando la instrucción de salto incondicional **JMP**.

Direccionamiento del SP (*stack pointer*), con desplazamiento de 8 *bits*

Hay 24 instrucciones de tres *bytes* que manejan el apuntador de la pila con *offset* de 8 *bits.* Este modo de direccionamiento sobre el SP nos permite movimientos ágiles dentro de 256 posiciones de la pila, la CPU encuentra la dirección efectiva del operando al sumar el registro SP de 16 *bits* con el *byte* que sigue al operando.

En este modo el SP se puede usar como otro registro índice de la CPU, siempre y cuando las interrupciones estén inhabilitadas.

En la **figura 21.19** se muestra un ejemplo de código de programa que ilustra el modo de direccionamiento del SP con *offset* de 8 *bits*, por medio de las instrucciones **STA** (guardar el acumulador en memoria) y **DBNZ** (decremento de la posición de memoria y salto, si no es igual a cero); en este programa se guarda el valor $20 en la posición $10, el SP se hace igual a $10+$FF=$10F, y se efectúa el decremento de la posición hasta cero.

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
450100		LDHX	#$0100	
94		TXS		;Reinicia el apuntador ;de la pila ;en $00FF
A620		LDA	#$20	;A = $20
9EE710		STA	$10,SP	;$10F = $20
9E6B10FC	LP	DBNZ	$10,SP,L P	;Offset de 8-bits ;decrementa el ;contenido de la ;posición $10F ;hasta llegar a cero

Figura 21.19 *Ejemplo del modo de direccionamiento del SP con offset de 8 bits*

Direccionamiento del SP, con desplazamiento de 16 *bits*

La familia del MCU 68HC08 tiene disponibles 14 instrucciones para el direccionamiento del SP con *offset* de 16 *bits.*

Todas las instrucciones en este modo son de cuatro *bytes* y se usan para acceder a datos de la pila ubicados en cualquier área de memoria; en este caso la CPU suma el contenido del SP con los dos *bytes* que preceden al código de operación, para determinar la dirección efectiva del operando.

En la **figura 21.20** se muestra un programa de ejemplo que ilustra el modo de direccionamiento del SP con *offset* de 16 *bits*, usando la instrucción LDA (carga del acumulador desde memoria); en este programa el acumulador se carga con el contenido de la posición de memoria $250 (SP=$250+$FF=$34F).

Direccionamiento relativo

El modo de direccionamiento relativo, en esta familia de microcontroladores, se maneja con 27 instrucciones. En todas las instrucciones de salto condicional, la CPU usa el direccionamiento relativo para determinar la dirección

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
450100		LDHX	#$0100	
94		TXS		;Reinicia el apuntador ;de la pila ;en $00FF
9ED60250		LDA	$0250,SP	;Offset de 16-bits ;Carga el registro A ;con el contenido ;de $34F

Figura 21.20 *Ejemplo del modo de direccionamiento del SP con offset de 16 bits*

efectiva (*effective address*; EA) del salto, ésta encuentra el destino del salto condicional al sumar el *byte* signado (en complemento a dos entre -128 a +127) que sigue al código de operación, con el contenido del contador de programa (PC).

Si la condición del salto se cumple, el PC se carga con la EA, de lo contrario, la CPU continúa con la ejecución normal de la próxima instrucción apuntada por el PC.

En la **figura 21.21** se muestra un programa de ejemplo que usa dos instrucciones del modo de direccionamiento relativo: **BLT** (salto si es menor que, operación signada), y **BRA** (salto incondicional); en éste, el contenido del acumulador se compara con -2, y ya que el número #1 es mayor que -2, el salto a TAG nunca ocurrirá.

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
A601	TAG	LDA	#1	;A = 1
A1FE		CMP	#-2	;Compara con -2
91FA		BLT	TAG	;Salta si el valor del ;registro A ;es menor que -2
20FE	HERE	BRA	HERE	;Salta siempre

Figura 21.21 *Ejemplos de instrucciones del modo de direccionamiento relativo*

CÓDIGO DE MÁQUINA	TIEMPO DE EJECUCIÓN	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
* Movimiento de datos con el registro acumulador				
B750	(2 ciclos)	PSHA		;Guarda el valor ;actual del registro A
A622	(2 ciclos)	LDA	#$22	;A = $22
B7F0	(3 ciclos)	STA	$F0	;Guarda el dato $22 ;dentro de la ;posición $F0
B650	(2 ciclos)	PULA		;Recupera el valor ;previamente guardado ;del registro A
	9 ciclos			
* Movimiento de datos sin el registro acumulador				
6E22F0	(4 ciclos)	MOV	#$22,$F0	;Ubicación del dato ;$22 dentro de la ;posición de memoria ;$F0

Figura 21.22 *Ejemplo del modo de direccionamiento inmediato a directo*

Direccionamiento de memoria a memoria, inmediato a directo

En este modo de direccionamiento, el movimiento de un operando inmediato (inmd) a una posición directa en memoria se hace por medio de la instrucción de tres *bytes* y cuatro ciclos de ejecución **MOV inmd**; este modo se usa con frecuencia para inicializar registros y variables en la página cero. La CPU guarda el primer *byte* que sigue al código de operación, en la dirección de la página directa que está indicada por el segundo *byte* de la instrucción.

En la **figura 21.22** se muestran dos segmentos de programa que ilustran la disminución significativa, de 9 a 4 ciclos, que resulta cuando no se usa el acumulador; ésto se ilustra con dos operaciones similares de transferencia inmediata a directa de este modo.

Direccionamiento de memoria a memoria, directo a directo

En este modo de direccionamiento, el movimiento de un operando directo en memoria a una posición directa en memoria, se hace por medio de la instrucción de tres *bytes* y cinco ciclos de ejecución **MOV dd**; este modo se usa para efectuar transferencias de datos dentro de la página cero. El operando que se encuentra en la posición de la página cero, indicada por el primer *byte* que sigue al código de operación, se guarda en la posición de la página cero indicada por el segundo *byte*.

CÓDIGO DE MÁQUINA	TIEMPO DE EJECUCIÓN	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
* Movimiento de datos con el acumulador				
B750	(2 ciclos)	PSHA		;Guarda el valor de A ;en la pila
B6F0	(3 ciclos)	LDA	$F0	;Captura el contenido ;de la posición $F0
B7F1	(3 ciclos)	STA	$F1	;$F1 = $F0
B650	(2 ciclos)	PULA		;Recupera el valor ;del registro A ;guardado en la pila
	10 ciclos			
* Movimiento de datos sin el acumulador				
4EF0F1	(5 ciclos)	MOV	$F0,$F1	;Mueve el contenido ;de la posición de ;memoria $F0 a la ;posición $F1

Figura 21.23 *Ejemplo del modo de direccionamiento directo a directo*

En la **figura 21.23** se muestran dos segmentos de programa que efectúan la misma operación de transferencia de datos directo a directo sin y con el acumulador, y la disminución correspondiente en el número de ciclos de ejecución que resulta en cada caso (de 10 a 5 ciclos).

Direccionamiento de memoria a memoria, indexado a directo con incremento

El movimiento de memoria a memoria indexado a directo y con incremento posterior, se efectúa con la instrucción de dos *bytes* y cuatro ciclos de ejecución **MOV ix+d**, y se destina generalmente para transferir tablas de datos apuntadas por el registro índice, hasta los registros de la página directa (la tabla puede tener cualquier tamaño y estar localizada en cualquier área dentro de los 64 Kbytes de memoria). La CPU guarda el operando apuntado por el registro H:X en la página cero, en la dirección que se indica en el *byte* que sigue al código de operación de la instrucción, e incrementa el registro índice después de cada movimiento.

CÓDIGO DE MÁQUINA	ETIQUETA	MNEMÓNICO DE LA OPERACIÓN	OPERANDO	COMENTARIOS
7E 18		MOV	X+, SCDR	;Mueve un nuevo byte ;al registro de ;datos SCI, ;Borra a TDRE y ;posteriormente ;incrementa el ;registro índice H:X.
65 00 64		CPHX	#TX_B + SIZE	;¿Ha sido finalizado ;el buffer circular?
23 03		BLS	NOLOOP	;Si no, continue
45 00 54		LDHX	#TX_B	;de lo contrario ;reinicie para ;arrancar el buffer
35 50	NOLOOP	STHX	PTR_OUT	;Guarda el nuevo ;valor del apuntador
80		RTI		;Retorno de ;interrupción

Figura 21.24 *Ejemplo del modo de direccionamiento indexado a directo con incremento*

Este modo de direccionamiento es bastante útil para transferir un *buffer* de datos de la memoria RAM a un registro de transmisión serial, tal como se muestra en la **figura 21.24**, en donde aparece un segmento de una rutina de atención a la interrupción de transmisión serial SCI.

Direccionamiento de memoria a memoria, directo a indexado con incremento

El movimiento de memoria a memoria, indexado a directo y con incremento posterior se efectúa con la instrucción de dos *bytes* y cuatro ciclos de ejecución **MOV dix+**, y se usa comúnmente para llenar tablas con registros de la página cero. El operando apuntado en la página cero, por el *byte* siguiente al código de operación, se guarda en la dirección apuntada por el registro H:X; y la CPU incrementa el registro índice después del movimiento. Como un ejemplo de este modo de direccionamiento se muestra el programa de servicio a la interrupción SCI en la **figura 21.25**.

Direccionamiento indexado con incremento posterior

Las instrucciones CBEQ y MOV son las dos instrucciones de dos *bytes* en las que se soporta el modo de direccionamiento indexado sin *offset* y con incremento posterior. En el registro H se guarda el *byte* de la parte alta de la direc-

```
CÓDIGO DE MÁQUINA   ETIQUETA   MNEMÓNICO DE LA OPERACIÓN   OPERANDO      COMENTARIOS
                    SIZE        EQU                     16            ;Tamaño del buffer
                                                                      ;circular

                    SCSR1       EQU                     $16           ;Reg.1 de estado SCI
                    SCDR        EQU                     $18           ;Registro receptor SCI
                                ORG                     $70
                    PTR_OUT     RMB                     2             ;Apuntador de datos
                                                                      ;de salida del
                                                                      ;buffer circular
                    PTR_IN      RMB                     2             ;Apuntador de datos
                                                                      ;de entrada del
                                                                      ;buffer circular
                    RX_B        RMB                     SIZE          ;Buffer circular

* Rutina de interrupción del registro receptor de datos SCI

                                ORG                     $6E00
55 72               RX_INT      LDHX                    PTR_IN        ;Carga el apuntador
B6 16                           LDA                     SCSR1         ;Lectura de SCSR1
                                                                      ;como parte del reset
5E 18                           MOV                     SCDR ,X+      ;Mueve un nuevo byte
                                                                      ;a SC1, borra a RDRF
                                                                      ;e incrementa H:X
65 00 64                        CPHX                    #RX_B + SIZE  ;¿Ha finalizado el
                                                                      ;buffer circular?
23 03                           BLS                     NOLOOP        ;Si no, continue
45 00 54                        LDHX                    #RX_B         ;De lo contrario
                                                                      ;inicialica desde el
                                                                      ;comienzo del buffer
35 52               NOLOOP      STHX                    PTR_IN        ;Guarda el nuevo
                                                                      ;valor del apuntador
80                              RTI                                   ;Retorna de la
                                                                      ;interrupción
```

Figura 21.25 *Ejemplo del modo de direccionamiento directo a indexado con incremento*

```
CÓDIGO DE MÁQUINA   ETIQUETA   MNEMÓNICO DE LA OPERACIÓN   OPERANDO      COMENTARIOS
A6FF                            LDA                     #$FF          ;A = $FF
B710                            STA                     $10           ;LOC $10 = $FF
4E1060                          MOV                     $10, $60      ;LOC $60 = $FF
5F                              CLRX                                  ;X = cero

* Compara el contenido del registro A con el contenido de la
* posición de memoria apuntada por el registro H:X y salta a TAG
* cuando son iguales

7102                LOOP        CBEQ                    X+,TAG        ;SIN OFFSET
20FC                            BRA                     LOOP          ;Evalua la próxima
                                                                      ;posición
5F                  TAG         CLRX                                  ;X = cero
```

Figura 21.26 *Ejemplo del modo de direccionamiento indexado con incremento*

```
CÓDIGO DE MÁQUINA   ETIQUETA   MNEMÓNICO DE LA OPERACIÓN   OPERANDO      COMENTARIOS
A6FF                            LDA                     #$FF          ;A = $FF
B710                            STA                     $10           ;LOC $10 = $FF
4E1060                          MOV                     $10, $60      ;LOC $60 = $FF
5F                              CLRX                                  ;X = cero

* Compara el contenido del registro A con el contenido de la
* posición de memoria apuntada por el registro H:X + $50 y salta
* a TG1 cuando son iguales

615002              LOOP2       CBEQ                    $50,X+,TG1    ;OFFSET DE 8-BITS
20FB                            BRA                     LOOP2         ;Evalúa la próxima
                                                                      ;posición
20FE                TG1         BRA                     TG1           ;Finaliza
```

Figura 21.27 *Ejemplo del modo de direccionamiento indexado con offset e incremento*

ción del operando, mientras que el *byte* más bajo se guarda en X, y, la suma, es la dirección condicional del operando.

Este modo de direccionamiento se usa con frecuencia para buscar datos dentro de tablas; la **figura 21.26** es un segmento de programa que usa la instrucción CBEQ (compare y salte si es igual, indexada) para ilustrar este tipo de direccionamiento.

Direccionamiento indexado con *offset* de 8 *bits* e incremento posterior

Con las mismas instrucciones, **CBEQ** y **MOV**, pero usando el *offset*, se puede manejar el modo de direccionamiento indexado con desplazamiento de 8 *bits* e incremento posterior. Por medio de estas instrucciones de tres *bytes* se puede acceder a cualquier operando con posición variable, e incrementar el registro índice después de cada operación, y de la misma forma que en el direccionamiento sin *offset*, la dirección condicional del operando se obtiene de la suma de los registros índice H y X.

Este modo de direccionamiento se usa también para localizar datos dentro de tablas; la **figura 21.27** es un segmento de programa que usa la instrucción CBEQ (compare y salte si es igual, indexada y con *offset*) para ilustrar este tipo de direccionamiento.

LECCIÓN 22

LAS INTERRUPCIONES EN LOS MICROCONTROLADORES 68HC08

La conexión y comunicación de un microcontrolador con microprocesadores, con dispositivos electrónicos, y con otros microcontroladores de diferente tecnología y velocidad de procesamiento, se hace gracias a las interrupciones.

Por medio de cualquier solicitud de interrupción, de origen interno o externo, la CPU detiene temporalmente la ejecución del programa principal mientras ejecuta el programa de servicio a la interrupción.

Una característica importante de los microcontroladores, debida a las interrupciones, es la flexibilidad en la conexión con dispositivos externos de cualquier tecnología y rango, así como también la facilidad para integrarse con otros sistemas de características diferentes.

Bajo condiciones de operación normal, la CPU08 de los microcontroladores ejecuta secuencialmente cada una de las instrucciones del programa principal; pero, con frecuencia, es necesario abandonar el programa principal y ejecutar, un conjunto de instrucciones determinadas y localizadas en un área de memoria conocida, en respuesta a la solicitud de interrupción hecha desde uno o varios dispositivos periféricos externos e internos. Generalmente, la solicitud para la ejecución del programa, o rutina de servicio a la interrupción, llegará al MCU fuera de sincronización con los tiempos de procesamiento del código de operación de la CPU, y su solicitud deberá atenderse en el menor tiempo posible.

En los microcontroladores Motorola con CPU08 se pueden procesar hasta 128 interrupciones de manera independiente; dentro de éstas se incluyen: las fuentes externas como el $\overline{\text{RST}}$ y la interrupción $\overline{\text{IRQ1}}$; y las fuentes internas como las interrupciones por *software* SWI.

Casi todas las interrupciones pueden ser enmascarables, y solo algunas pocas no, como por ejemplo el *reset*. Las enmascarables se manejan desde el sistema periférico interno destinado para tal propósito, por medio del *bit* que controla la máscara global de las interrupciones (***Bit* I**), el cual está localizado en la tercera posición del registro del código de condiciones (**CCR**). Todas las interrupciones se habilitan cuando este *bit* está en cero lógico.

Procesamiento de interrupciones en la CPU08

La solicitud efectuada de parte de cualquier fuente de interrupción, excepto el *reset*, se atiende de acuerdo con los resultados que se obtienen tras la ejecución de una secuencia de cuatro tareas por parte del módulo de interrupciones del MCU; ellas son:

- Reconocimiento de la fuente de interrupción
- Evaluación del nivel de prioridad (arbitraje)
- Almacenamiento del estado actual del microcontrolador
- Búsqueda del vector de la interrupción

En la **figura 22.1** se muestran las tareas que la CPU08 ejecuta durante el procesamiento de una interrupción; en ella se incluyen, además de la secuencia de operaciones, los tiempos del bus interno en que cada una se desarrolla.

Nota:
El *reset* externo o interno, es la interrupción principal del microcontrolador y se usa para inicializar los registros principales; no es enmascarable, goza del más alto nivel de prioridad y por este motivo la atención a su solicitud de interrupción es la más rápida. Esto se logra porque el módulo de interrupciones, al procesar el *reset*, sólo efectúa dos de las cuatro tareas que se requieren típicamente para el procesamiento de las interrupciones generales, que son: el reconocimiento de la fuente de interrupción y la búsqueda del vector correspondiente.

Reconocimiento de la fuente de interrupción

En el caso del *reset*, el reconocimiento no es sincrónico con el procesamiento de la CPU, y es inmediato; mientras que la salida de este estado de interrupción siempre estará sincronizada con los ciclos de ejecución del bus interno.

Las demás interrupciones siempre serán reconocidas por la CPU08 solamente durante el último ciclo, de ejecución de cada instrucción, y cualquier solicitud de interrupción que se haga en este ciclo, será ignorada, y sólo se volverá a considerar hasta el último ciclo de la próxima instrucción; con lo que queda claro que la CPU08 nunca detiene la ejecución de una instrucción

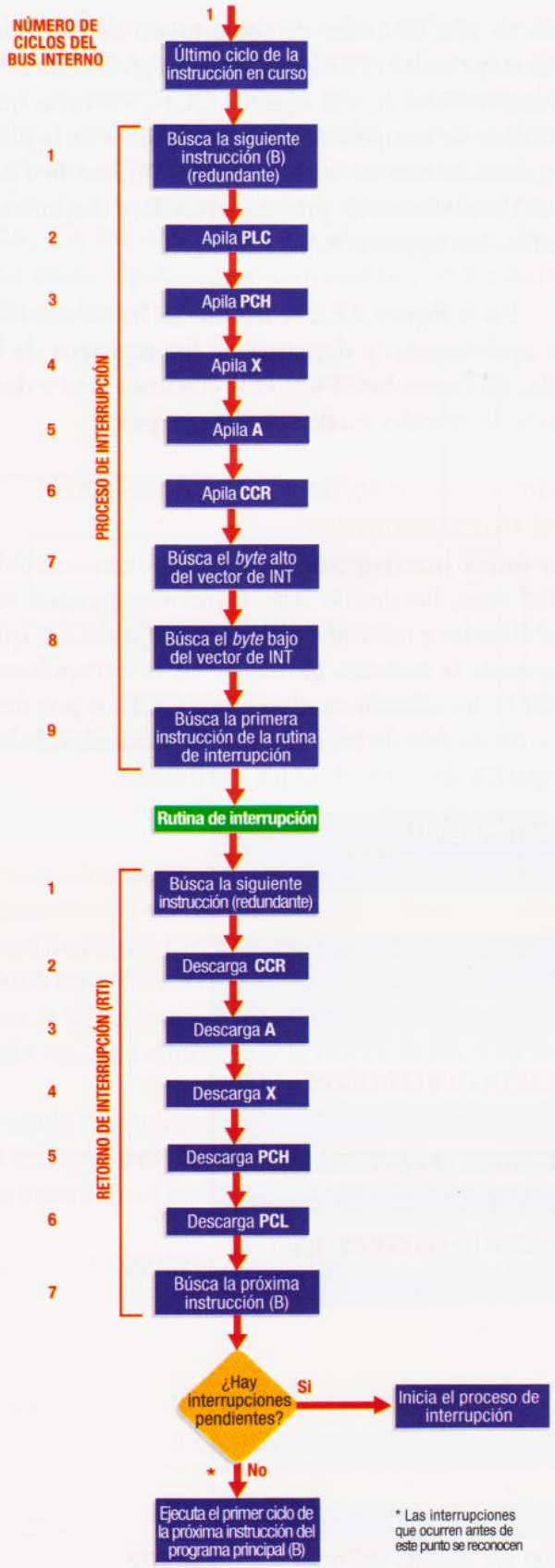


Figura 22.1 *Secuencia de operaciones del procesamiento de una interrupción*

para atender una rutina de interrupción normal (solamente cuando se está usando el módulo DMA, éste puede suspender la operación de una instrucción en la CPU para atender a un dispositivo periférico).

Al reconocer una interrupción, la CPU08 carga el registro decodificador de instrucciones con el código de operación asignado por *software* que le corresponde a dicha interrupción (**SWIx**), en lugar de cargar el código de operación de la que debió haber sido la siguiente instrucción en el programa. El contador del programa (**PC**) es el otro registro que se altera cuando se reconoce una interrupción, a éste se le resta uno antes de ser guardado en la pila, ya que, debido al mecanismo de prebúsqueda del código de operación de la CPU, siempre está apuntando a la dirección de la próxima instrucción más uno.

Evaluación del nivel de prioridad (arbitraje)

Todas las interrupciones de *reset* por defecto tienen la misma y la más alta prioridad, por lo tanto para ellas la CPU08 no efectúa la operación de evaluación del nivel de prioridad, que se lleva a cabo durante el procesamiento normal de las otras interrupciones.

Todas las interrupciones que llegan al microcontrolador simultáneamente, son sometidas al proceso de evaluación del nivel de prioridad en el módulo del sistema de integración (**SIM**), y éste, se basa en el orden de prioridades de los vectores que se muestran en la **tabla 22.1**. Como resultado, el módulo SIM obtiene un valor constante que en su momento utiliza para localizar la dirección efectiva del vector de interrupción que debe ser buscado, para poder atender la rutina de servicio.

Cuando la CPU reconoce una fuente de interrupción, el correspondiente *bit* de señalización (*flag* o bandera) del registro de estado lo indica al ponerse en uno lógico; y, el diseñador

del programa es el responsable de borrar (o colocar en cero lógico) el señalizador, luego de ejecutar apropiadamente la rutina de servicio solicitada por la interrupción.

Dirección	Interrupción	Prioridad
FFFE	*Reset*	1
FFFC	SWI	2
FFFA	IREQ[0]	3
:	:	:
FF02	IREQ[124]	127
FF00	IREQ[125]	128

Tabla 22.1 *Tabla de niveles de prioridad de los vectores de las interrupciones de la CPU08*

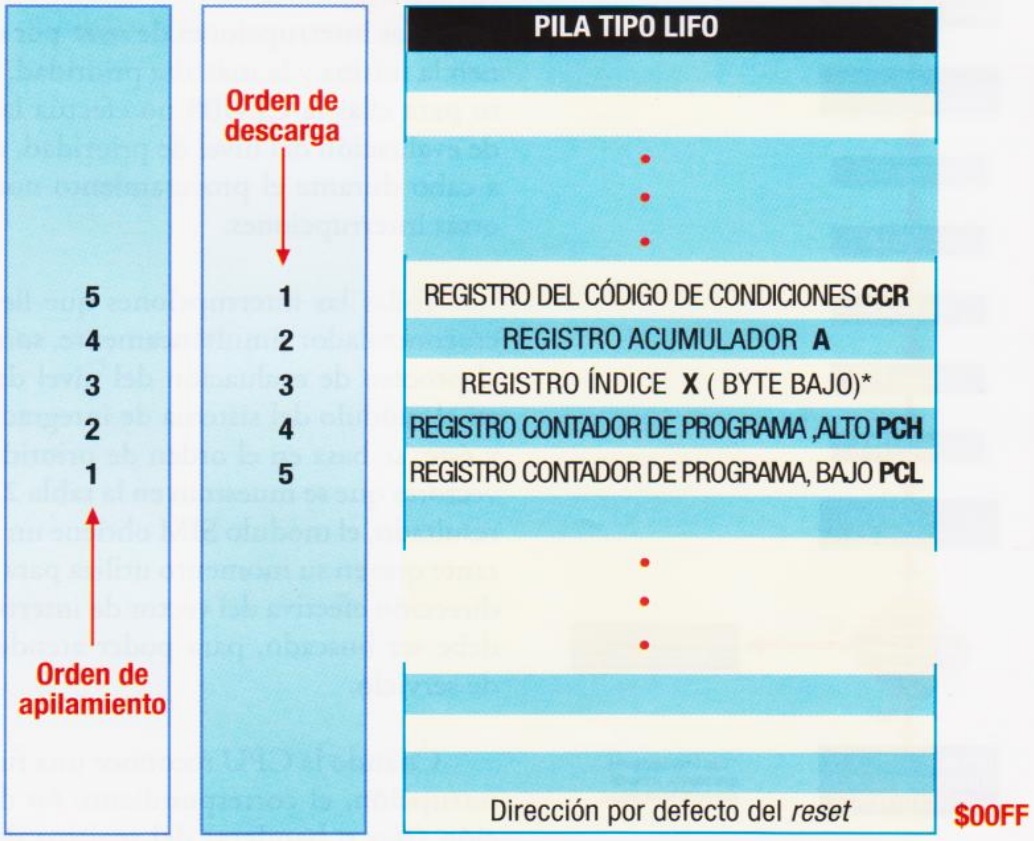
Almacenamiento del estado actual del microcontrolador

Para preservar el estado del MCU después de retornar de una rutina de servicio de interrupción, la CPU08 almacena en la pila los principales registros antes de atender la interrupción, y los descarga al salir de ella. El orden de apilamiento de los registros es: primero el contador **PC**, luego el índice **X**, el acumulador **A**, y el registro **CCR**; mientras que el orden de recuperación de los registros de la pila, de acuerdo con su estructura LIFO (*Last In First Out)*, es al contrario: primero el **CCR**, y después,en orden, los registros **A**, **X**, y **PC**.

En la **figura 22.2** se muestran las secuencias de apilamiento y descarga de los registros de la pila, tal como la CPU08 las ejecuta antes y después de atender cualquier interrupción.

Enmascaramiento y anidamiento de las interrupciones

La única interrupción que no es enmascarable es el *reset*, las demás interrupciones pueden ser habilitadas e inhabilitadas por medio del *bit* que controla la máscara global de las interrupciones (***Bit* I**), localizado en el registro **CCR**; o por medio de los *bits* de las máscaras individuales de los registros de control de los periféricos.



* El byte alto del registro índice H no se apila

Figura 22.2 *Orden de apilamiento y descarga de los registros principales en la pila*

Para permitir la ejecución anidada de las interrupciones en espera, la modificación del *bit* I se efectúa al menos un ciclo antes del último ciclo de la instrucción u operación, con esto se garantiza que su estado está ya definido antes de ejecutar la próxima instrucción. En los tres ejemplos siguientes se ilustran casos típicos de reconocimiento y ejecución de interrupciones.

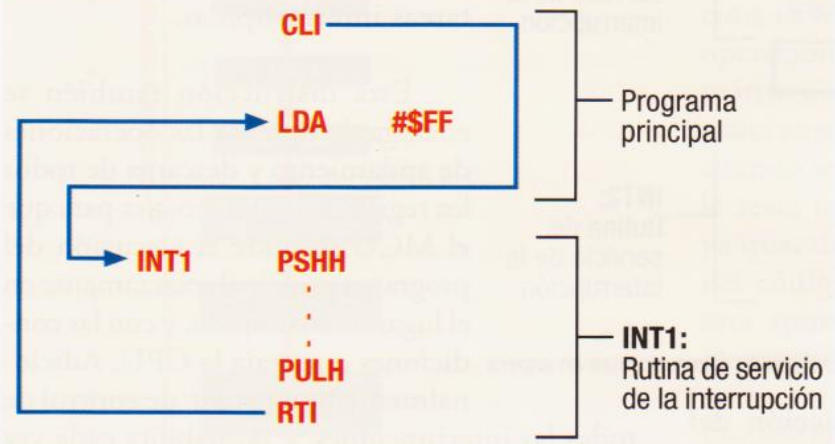


Figura 22.3 *Ejecución de interrupciones*

Como primer ejemplo, en la **figura 22.3** se muestra que, si la interrupción se reconoce durante la ejecución de la instrucción CLI, la CPU08 saltará a ejecutar la rutina de servicio de la interrupción, y al terminar, regresará al programa principal a ejecutar la siguiente instrucción LDA.

En el segundo ejemplo, de la **figura 22.4**, se muestra la ejecución anidada de dos rutinas de servicio de interrupciones INT1 e INT2; en este caso las solicitudes de interrupción pueden llegar al mismo tiempo, o la segunda (INT2) llegar durante la ejecución de la primera (INT1). En cualquier caso la interrupción INT2 será atendida solamente al terminar la rutina de servicio de la primera.

También es importante anotar que, durante cada instrucción de retorno RTI, la CPU08 busca

PARA SABER MÁS:
Apilamiento compatible

Para mantener la compatibilidad de los programas con la familia de MCU 68HC05, la CPU de la familia HC08 no apila el contenido del registro índice alto H durante el proceso de atención a las interrupciones, y por lo tanto, el programador no debe olvidarse de guardar y recuperar este registro al entrar y salir respectivamente de la rutina de servicio de la interrupción.

Usando las instrucciones **PSHH** y **PULH** para el manejo del área de la pila, se puede apilar y descargar el contenido del registro **H** para preservarlo durante el acceso a una rutina de interrupción, tal como se ilustra en el siguiente segmento de código.

```
IRQINT    PSHH     ; Se guarda en la pila el contenido del registro H
          .
          .
          .
RUTINA DE SERVICIO
A LA INTERRUPCIÓN
          .
          .
          .
          PULH     ; Se descarga de la pila el contenido del registro H
          RTI      ; Retorna al programa principal
```

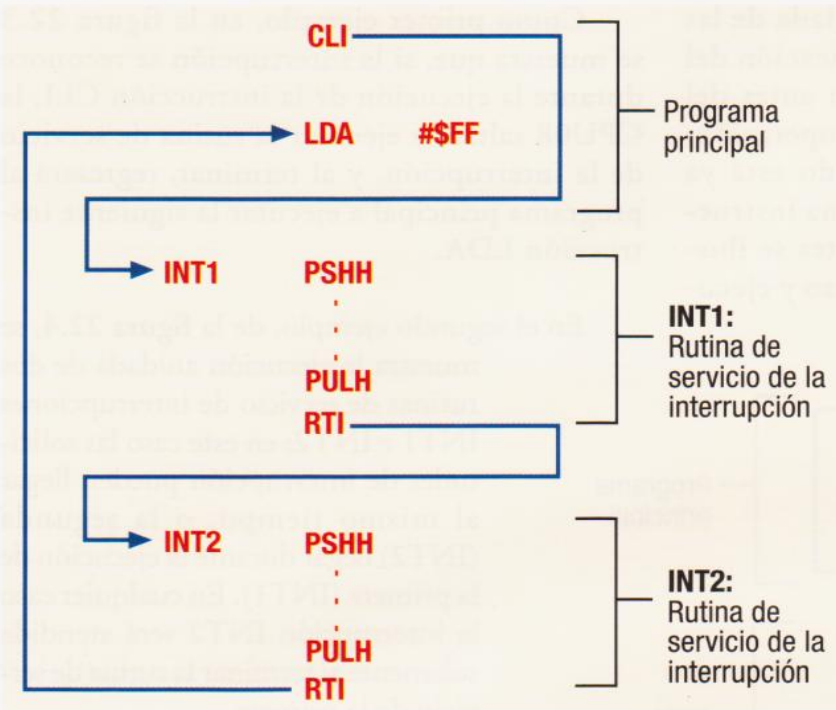


Figura 22.4 *Ejecución anidada de interrupciones en espera*

la dirección de la siguiente instrucción del programa principal (LDA), aunque en el caso del retorno de la INT1, sólo es una acción redundante.

El tercer ejemplo, de la **figura 22.5**, nos muestra que, si una fuente de interrupción se detecta durante la ejecución de la instrucción CLI, ésta será atendida antes de colocar en uno lógico el *bit* I del registro **CCR**, e inhabilitar o poner la máscara sobre todas las interrupciones.

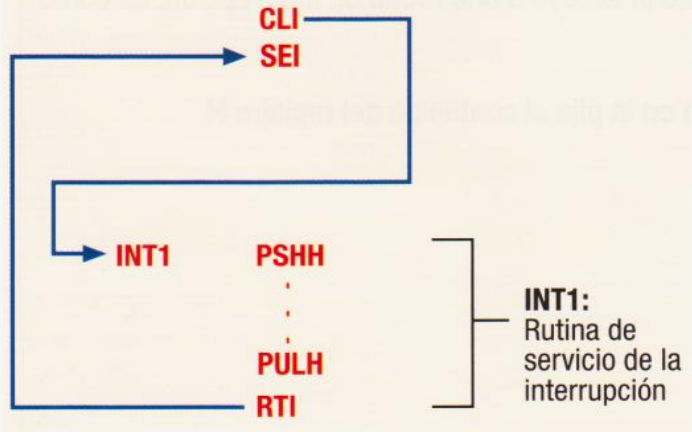


Figura 22.5 *Ejecución y reconocimiento de la interrupción durante la ejecución de una instrucción*

Retorno de la interrupción (RTI)

Al terminar la ejecución de la rutina de servicio de la interrupción, y aún dentro de ésta, se debe ubicar la instrucción de retorno de interrupción **RTI**; ésta se encarga de finalizar la ejecución de la rutina de servicio y devolver el apuntador de programa al programa principal, para continuar con las tareas interrumpidas.

Esta instrucción también se encarga de ordenar las operaciones de apilamiento y descarga de todos los registros fundamentales para que el MCU reanude la ejecución del programa principal, exactamente en el lugar interrumpido, y con las condiciones que traía la CPU. Adicionalmente borra el *bit* de control de todas las interrupciones, y las habilita cada vez que descarga el registro **CCR** de la pila.

Módulo de interrupción externa (IRQ)

El módulo de interrupción externa **IRQ**, maneja y nos permite controlar la operación de ésta por medio de sus registros. Este módulo completo se muestra en el diagrama de bloques de la **figura 22.6**; y las características con las que cuenta en los MCU de la familia HC08, son:

- Un pin de entrada externa para la interrupción $\overline{\text{IRQ1}}$
- *Bits* de control para la entrada $\overline{\text{IRQ1}}$
- Un *buffer* de entrada con histéresis
- Entrada con sensibilidad programable por flanco, o por flanco y nivel
- Reconocimiento automático
- Selector programable de resistencia interna de *pull-up*

Al aplicar una señal con nivel bajo (un cero lógico) en el pin IRQ1 del microcontrolador, se activa la solicitud de servicio de la interrupción externa en la CPU y se almacena en el *latch* IRQ1.

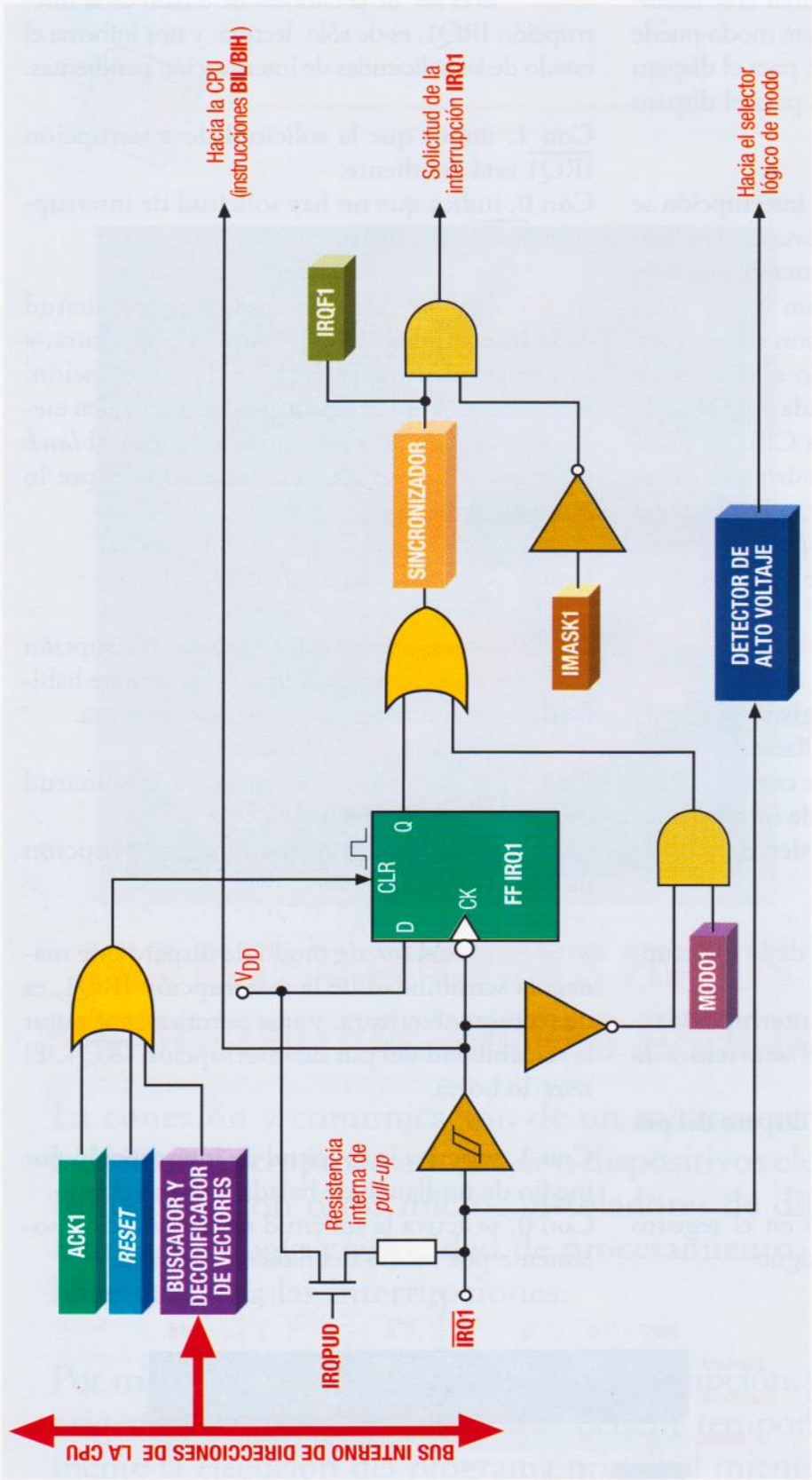


Figura 22.6 *Diagrama de bloques del módulo de interrupción externa IRQ*

Una vez reconocida la interrupción, el *latch* permanecerá en uno hasta que ocurra alguna de las siguientes operaciones:

1. La búsqueda de un vector: el reconocimiento de otra solicitud de interrupción originará la búsqueda del vector correspondiente, y durante este ciclo, el *latch* IRQ será borrado.

2. Un borrado por *software*: éste puede usarse para borrar el estado memorizado de la interrupción, si se accede el *bit* de reconocimiento (**ACK1**) que hace parte del registro de estado y control de la interrupción **INTSCR**. Al escribir un uno lógico en éste, se borra el *latch* IRQ1.

3. La ejecución de un *reset*: esta operación tiene dentro de sus tareas el borrado del *latch* de la interrupción externa **IRQ1**.

4. El pin de entrada $\overline{\text{IRQ1}}$ se inactiva si se coloca en nivel lógico uno, y si previamente se ha configurado su sensibilidad por nivel; bajo esta condición, el *latch* IRQ1 será borrado.

Pin de interrupción externa ($\overline{\text{IRQ1}}$)

La entrada de la interrupción externa $\overline{\text{IRQ1}}$ es activa en nivel bajo, y por defecto responde a los flancos de bajada. Sin embargo, por medio del *bit* de control del modo de disparo **MODE1**, propio del

registro **INTSCR**, se puede programar la sensibilidad del disparo en dicha entrada; este modo puede configurarse en una de dos formas: para el disparo por flanco de bajada solamente, o para el disparo por flanco de bajada y nivel bajo.

Cuando el disparo del pin de interrupción se configura por flanco de bajada solamente, la solicitud de interrupción en la CPU se mantiene activa hasta que ocurre la búsqueda de un vector, o un borrado por *software*, ó, la ejecución de un *reset*. Pero, cuando se haya configurado el disparo de la interrupción por flanco de bajada y nivel bajo, la solicitud de interrupción en la CPU se mantendrá activa hasta que ocurra la búsqueda de un vector, o un borrado por *software*, o la ejecución de un *reset*, ó, el pin de la interrupción regrese a su condición inactiva (uno lógico).

Registro de estado y control de la interrupción IRQ (INTSCR)

En la **figura 22.7** se muestra el registro de estado y control de la interrupción IRQ, llamado **INTSCR**. Por medio de este registro se controla y supervisa la operación del módulo de interrupción externa IRQ; a través de él se pueden desarrollar las siguientes funciones:

1. Se vigila la bandera de estado de la interrupción **IRQF1**
2. Se borra el *latch* IRQ1 de la interrupción
3. Se enmascara o se inhibe el servicio a la interrupción
4. Se controla la sensibilidad del disparo del pin de interrupción $\overline{\text{IRQ1}}$

Los *bits* de control y estado en el registro **INTSCR** están definidos como sigue:

IRQF1: es el *bit* de la bandera de estado de la interrupción IRQ1, es de sólo lectura, y nos informa el estado de las solicitudes de interrupción pendientes.

Con **1**, indica que la solicitud de interrupción $\overline{\text{IRQ1}}$ está pendiente.
Con **0**, indica que no hay solicitud de interrupción $\overline{\text{IRQ1}}$ pendiente.

ACK1: es el *bit* de reconocimiento de solicitud de la interrupción IRQ1, es de sólo escritura, y nos permite borrar el *latch* de la interrupción. Este *bit* interruptor se usa con frecuencia en ejecuciones anidadas que requieren borrar el *latch* IRQ por *software*. El ciclo de *reset* siempre lo pone en cero.

Con **1**, se borra el *latch* de IRQ.

IMASK1: es el *bit* de máscara de la interrupción IRQ1; es de lectura y escritura, y nos permite habilitarla o inhabilitarla. Luego del *reset* se borra.

Con **1**, se inhabilita o enmascara la solicitud de interrupción de IRQ1.
Con **0**, se habilita la solicitud de interrupción de IRQ1.

MODE1: es el *bit* de modo de disparo que maneja la sensibilidad de la interrupción IRQ1, es de lectura y escritura, y nos permite configurar la sensibilidad del pin de interrupción $\overline{\text{IRQ1}}$. El *reset* lo borra.

Con **1**, se activa la solicitud de interrupción por medio de un flanco de bajada y un nivel bajo.
Con **0**, se activa la solicitud de interrupción solamente por medio del flanco de bajada.

Dirección	Nombre de registro		Bit 7	6	5	4	3	2	1	Bit 0
$001D	Registro de estado y control de la interrupción IRQ (INTSCR)	Lectura:	0	0	0	0	IRQF1	0	IMASK1	MODE1
		Escritura:						ACK1		
		Reset:	0	0	0	0	0	0	0	0

Sin establecer

Figura 22.7 *Registro de estado y control de la interrupción IRQ (INTSCR)*

LECCIÓN 23

LOS MÓDULOS ESPECIALES DEL 68HC08: COP, LVI Y KBI

Los módulos especiales, que vigilan la ejecución apropiada del *software* del microcontrolador y lo protegen durante las condiciones anormales del voltaje de alimentación, tales como los módulos COP (*computer operating properly*) y LVI (*low voltage inhibit*), se han incorporado dentro de la arquitectura de los microcontroladores para aumentar la confiabilidad de estos circuitos integrados.

El módulo que maneja las interrupciones dedicadas al manejo de teclados, y que facilita para el usuario la tarea de decodificar matrices de teclas, es el KBI (*keyboard interrupt module*); y es otra herramienta de mucho uso, ofrecida por esta familia de microcontroladores.

El módulo COP (*computer operating properly*)

Bajo el mismo concepto operativo del perro guardián (*watchdog*) que se ha venido incorporando en todas las arquitecturas modernas de los microprocesadores (MPU) y los microcontroladores (MCU) de otros fabricantes, la familia 68HC08 de Motorola nos ofrece un módulo que forma parte de la arquitectura de estos MCU, y que es el encargado de facilitar al programador la tarea de vigilar que la ejecución del programa residente en la memoria del dispositivo sea efectuada en forma cíclica y sin interrupciones.

El módulo de operación apropiado de la computadora (COP), ha sido diseñado con un contador de carrera libre que se encarga de generar una señal de *reset* interna en la CPU08, cada que se desborda su capacidad de conteo. EL COP ayuda al *software* a recuperar su flujo normal cada vez que el registro contador del programa PC se altera y el orden de ejecución del programa se pierde, ocasionando un comportamiento errado o inesperado del MCU. Dentro de las principales y más frecuentes causas que pueden hacer que la secuencia de ejecución de una rutina o programa se salga de su curso, se encuentran las siguientes:

- Diseño de algoritmos incorrectos
- Vectorización inapropiada
- Ejecución de instrucciones de salto fuera de rango
- Alteración de registros y banderas por ruido eléctrico inducido
- Modificación de registros durante la conexión y desconexión de la alimentación

Para evitar que el temporizador asociado al módulo COP, genere un ciclo de *reset*, se debe borrar periódicamente y antes de su desbordamiento; esto se debe hacer en uno o varios lu-

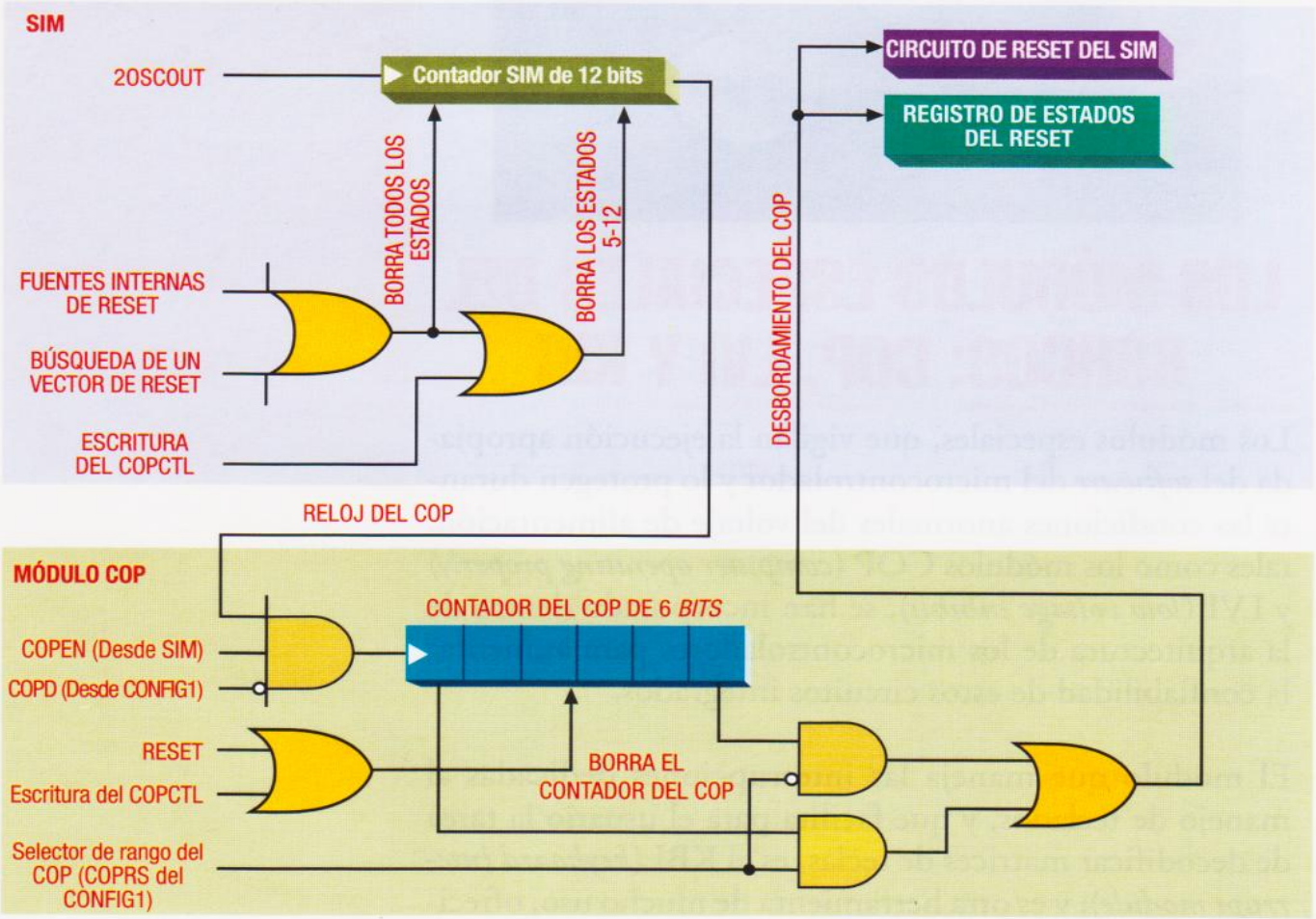


Figura 23.1 *Diagrama de bloques del módulo COP*

gares estratégicos del programa insertando instrucciones que coloquen en ceros el registro de este temporizador. La característica del temporizador de carrera libre garantiza que, luego de su borrado, la temporización de forma automática se inicia nuevamente.

Funciones del módulo COP

En la **figura 23.1** se muestra la estructura en bloques simplificada del módulo de vigilancia COP, en ella se puede observar que su contador posee un registro de 6 *bits*, y que esta precedido por el contador de 12 *bits* del módulo del sistema de integración (SIM). Si el programador no lo borra por *software*, el contador del COP se desbordará y activará un ciclo de *reset* fuera de sincronización después de haber ejecutado 2^{18} - 2^{4}, o 2^{13} - 2^{4} ciclos de reloj de la señal 2OSCOUT, y esto depende del estado del *bit* selector de rango del temporizador (COPRS), que se encuentra en el registro de configuración número uno.

Cuando el programador ha seleccionado la opción de desbordamiento a los 2^{18} - 2^{4} ciclos del reloj 2OSCOUT, se obtendrá el máximo periodo de temporización, lo cual significa que, por ejemplo, si usamos un cristal de 8MHz, el módulo generará un *reset* cada 32,766ms. Para inicializar de nuevo el temporizador del COP e impedir la ejecución del *reset*, es necesario borrar los registros de los contadores del COP (6 *bits*) y del SIM (12 *bits*) por medio de una o varias instrucciones ubicadas dentro del programa, antes de que se cumpla el tiempo de desbordamiento programado.

Las señales del módulo COP

En la figura 23.1 pueden verse las señales que intervienen en el módulo COP de la CPU08; éstas son: la entrada de reloj 2OSCOUT, la entrada de escritura COPCTL, la señal de *reset* a la conexión POR, el *reset* interno, la señal de búsqueda del vector de *reset*, la señal de inhabilitación del COP (COPD), y la señal selectora del rango del COP.

1. 2OSCOUT

Esta señal es la salida del módulo oscilador del MCU, y su frecuencia es igual a la del cristal de cuarzo elegido, o a la de la red RC establecida.

2. COPCTL

Por medio de este registro se pueden borrar los contadores de los módulos COP y SIM que intervienen en el temporizador de vigilancia del programa; para ello solamente se requiere escribir un valor cualquiera en el registro COPCTL, usando alguna de las instrucciones de escritura que forman parte del conjunto de instrucciones del microcontrolador.

En la **figura 23.2** se muestra el registro de control del módulo COP (COPCTL); éste se encuentra localizado en la posición de memoria $FFFF, y se sobreescribe en la parte baja del vector del *reset*. La escritura de un nuevo valor en este registro determina el inicio de un nuevo periodo de vigilancia por parte del temporizador del COP.

En cualquier caso, la lectura del registro COPCTL obligará a la CPU a devolver al programador el contenido de la parte más baja (el *byte* menos significativo) del vector del *reset*, que también está ubicado en la posición $FFFF

3. El reset a la conexión POR

Dentro del módulo SIM se encuentra el circuito que soporta el *reset* a la conexión (*Power-On Reset*, POR); en este caso, él se ocupa de generar una señal que borra el contador del SIM al transcurrir 4.096 periodos de la señal 2OSCOUT, luego de haber conectado la alimentación del MCU.

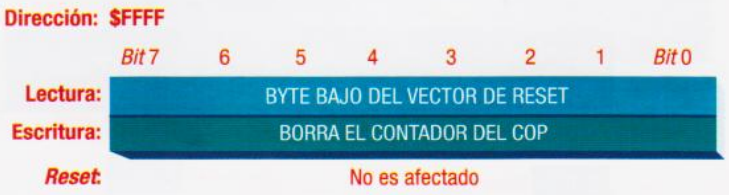


Figura 23.2 *Registro de control del módulo COP (COPCTL)*

4. El reset interno

Los dos contadores que intervienen en el tiempo de vigilancia del COP, son borrados tras el reconocimiento y la ejecución de cualquier ciclo de *reset* interno.

5. La búsqueda del vector de *reset*

Cuando la dirección de un vector de *reset* se coloca en el bus de direcciones, se inicia un ciclo de búsqueda del vector y con éste se borrará el contador del SIM.

6. La señal de inhabilitación del módulo COP

La entrada COPD es un reflejo del estado del *bit* 0 del registro de configuración número uno (CONFIG1), llamado también COPD, y por medio del cual se inhabilita la operación del temporizador del COP.

7. La entrada del selector de rango COPRS (COP *rate select*)

Por medio de esta entrada del módulo COP se determina el rango de los contadores, lo cual, de la mano con la frecuencia de operación, nos permite definir el máximo periodo para el temporizador vigilante. El *bit* 7 del registro CONFIG1 es el selector de rango del COP, llamado COPRS; y su estado se informa al módulo COP a través de esta entrada.

En la **figura 23.3** se presenta el registro de configuración número uno (CONFIG1) de la CPU08, y se describen las funciones de los dos *bits* que se involucran en el funcionamiento del módulo COP, COPRS y COPD.

Figura 23.3 *Primer registro de configuración (CONFIG1)*

COPRS: es el *bit* selector del rango del periodo del temporizador, y tras la ejecución de un ciclo de *reset* será borrado.

Con**1** lógico: el periodo programado será (2^{13} - 2^4) X 2OSCOUT ciclos.
Con **0** lógico: el periodo programado será (2^{18} - 2^4) X 2OSCOUT ciclos.

COPD: es el *bit* que nos permite habilitar o inhabilitar el módulo COP, y su estado será colocado en cero luego de la ejecución de un ciclo de *reset*.

Con **1** lógico: el módulo COP será inhabilitado.
Con **0** lógico: el módulo COP será habilitado.

Finalmente, vale la pena anotar el comportamiento del módulo COP durante algunos de los estados de operación más importantes del microcontrolador. En el modo monitor y en el modo de interrupción (BREAK), el COP es automáticamente inhabilitado cuando el pin $\overline{IRQ1}$ ó $\overline{RST}$ es puesto en un nivel de voltaje igual a $V_{DD} + V_{HI}$. Sin embargo, no ocurre lo mismo durante el estado de espera del MCU (modo WAIT); en este estado el COP no se detiene y debe usarse una rutina de interrupción para borrar periódicamente su contador.

En el otro modo de operación de bajo consumo, conocido como estado de paro del MCU (modo STOP), el contador del SIM se borra y la señal de entrada de reloj del módulo COP (2OSCOUT) se desconecta.

El módulo inhibidor por voltaje bajo LVI (*low voltage inhibitor*)

Cuando el voltaje de alimentación cae abruptamente, o el microcontrolador se somete a conexiones y desconexiones rápidas, es probable que los datos de registros y banderas de estado resulten alterados; esto sucede cuando la CPU08 pierde la alimentación durante el transcurso del proceso de ejecución de ciertas instrucciones, que involucran el movimiento de datos dentro del MCU.

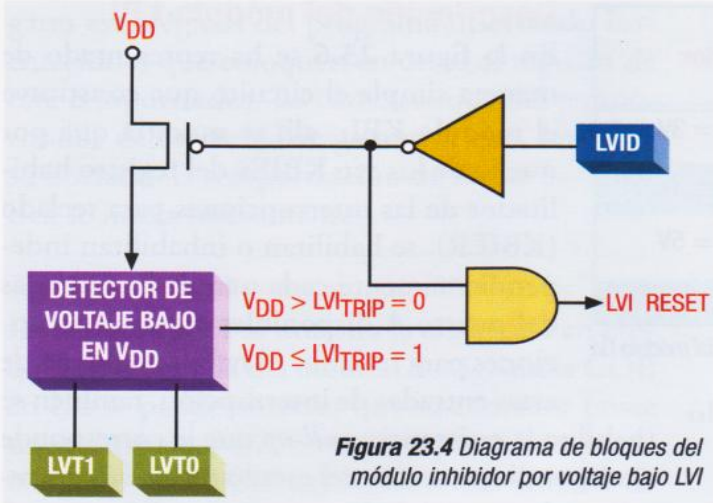


Figura 23.4 *Diagrama de bloques del módulo inhibidor por voltaje bajo LVI*

Con el propósito de evitar esto, los microcontroladores 68HC08 poseen un módulo que monitorea continuamente el nivel de voltaje presente en el pin de alimentación del microcontrolador V_{DD}, y genera un ciclo de *reset* cuando el voltaje cae por debajo del umbral definido para el inhibidor (*LVI trip voltage*). El circuito LVI puede ser habilitado o inhabilitado por el usuario, y el umbral de voltaje del inhibidor puede ser también seleccionado por el programador.

En la **figura 23.4** se muestra el diagrama en bloques de la estructura del circuito del módulo LVI, este modulo contiene un circuito especial (*bandgap*) de referencia y un comparador, como partes principales, y por medio de el *bit* de control LVID se puede habilitar o inhabilitar el circuito detector de voltaje de alimentación bajo, cuando se coloca en 0 ó 1 lógico, respectivamente.

Por medio de los *bits* de selección del voltaje de inhibición el programador le define a la CPU del microcontrolador cual será el nivel de voltaje al cual se protegerá, y generará la señal de *reset*; mientras que para activar el ciclo de *reset* el módulo LVI tiene la salida LVI *reset* que afecta directamente a la CPU.

Registros de control del módulo LVI (CONFIG2 y CONFIG1)

Todas las acciones de control sobre las funciones del módulo LVI se efectúan desde los registros CONFIG1 y CONFIG2; a continuación se describirán los *bits* que intervienen en él. En la figura 23.3 se muestra el registro CONFIG1, mientras que en la **figura 23.5** se puede apreciar el registro CONFIG2.

LVID: este *bit* pertenece al registro CONFIG1, se encuentra en la posición del *bit* 4, y es el interruptor de habilitación del inhibidor por voltaje bajo. Su condición siempre será cero luego del *reset*.

Con **1** lógico: el módulo LVI se inhabilita.
Con **0** lógico: el módulo LVI se habilita.

LVIT1 y LVIT0: estos dos *bits* pertenecen al registro CONFIG2, y por medio de ellos se puede programar el umbral de disparo del voltaje del módulo inhibidor (*LVI trip voltage*). Por medio de un ciclo normal de *reset* su estado no será afectado; sin embargo, cuando el *reset* es generado por el POR, ambos *bits* son borrados.

En la **tabla 23.1** se muestran las opciones de selección que pueden elegirse por medio de estos dos *bits*.

El módulo de interrupciones para teclado KBI

Como una herramienta de gran utilidad para el programador, y teniendo en cuenta el uso frecuente de los microcontroladores en las interfaces de teclados matriciales, los MCU de esta familia incluyen un conjunto de siete interrupciones externas que pueden ser enmascarables y están accesibles a través de los siete primeros pines del puerto A (PTA0 a PTA6).

Dirección: $001E

	Bit 7	6	5	4	3	2	1	*Bit* 0
Lectura: / **Escritura:**	IRQPUD	R	R	LVIT1	LVIT0	R	R	R
Reset:	0	0	0	No son afectados		0	0	0
POR:	0	0	0	0	0	0	0	0

R Reservado

Figura 23.5 *Segundo registro de configuración (CONFIG2)*

LVIT1	LVIT0	Voltaje de disparo	Comentarios
0	0	V_{LVR3} (2,4V)	Para V_{DD} = 3V
0	1	V_{LVR3} (2,4V)	Para V_{DD} = 3V
1	0	V_{LVR5} (4,0V)	Para V_{DD} = 5V
1	1	Reservado	

Tabla 23.1 *Tabla de selección del umbral de voltaje de inhibición del módulo LVI*

Las principales características del módulo KBI son :

1. Posee siete interrupciones activas en nivel bajo, con entradas enmascarables independientemente y también de manera global.

2. Tiene configuración por *software* de las resistencias de *pull-up* en cada entrada de interrupción.

3. El nivel de sensibilidad del disparo de la interrupción puede ser seleccionado por flanco solamente, o por flanco y nivel.

4. Todas las entradas de interrupción para teclado representan un medio eficaz para sacar al microcontrolador de cualquiera de los estados de bajo consumo STOP y WAIT.

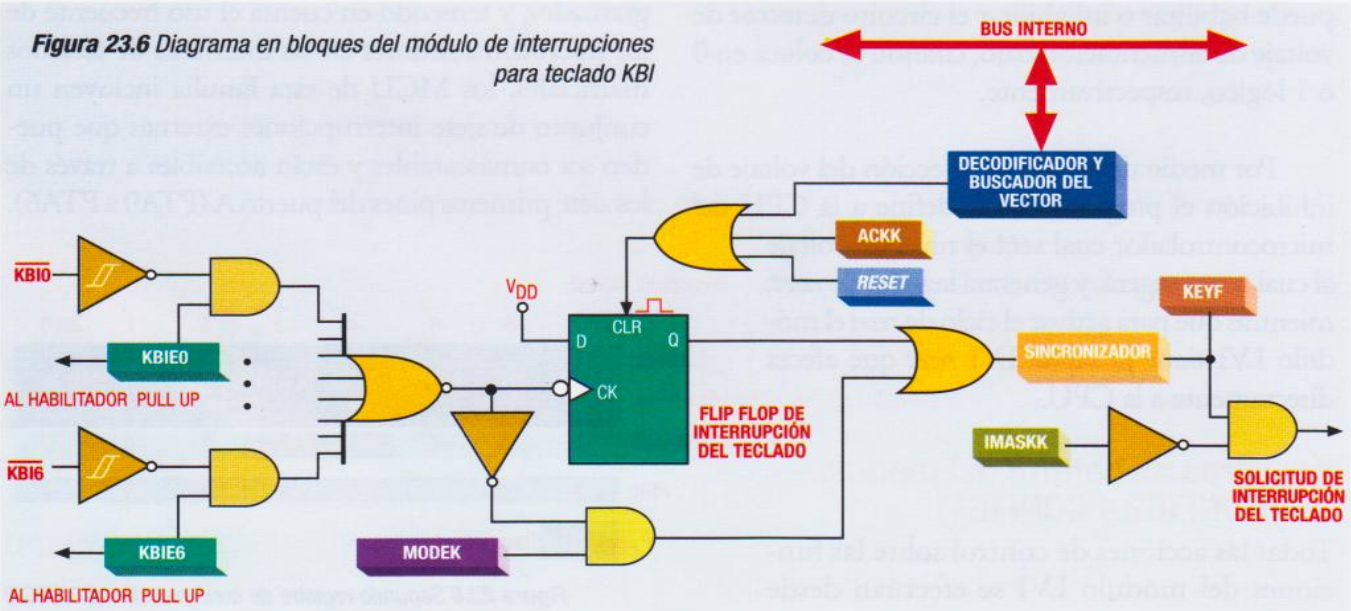
Descripción del módulo KBI

En la **figura 23.6** se ha representado de manera simple el circuito que constituye el módulo KBI; allí se muestra que por medio de los *bits* KBIEx del registro habilitador de las interrupciones para teclado (KBIER), se habilitan o inhabilitan independientemente cada una de las entradas del puerto A disponibles como interrupciones para teclado. Al habilitar alguna de estas entradas de interrupción, también se habilita la resistencia *pull-up* que le corresponde independientemente del estado de los *bits* del registro de control de las resistencias del puerto A (PTAPUEx).

Cuando una entrada de interrupción para teclado es habilitada, y recibe un cero lógico por el correspondiente pin, inmediatamente se memoriza e inicia un proceso de reconocimiento y solicitud de interrupción, y el modo de funcionamiento del KBI define que cada vez que uno o más de sus pines de entrada se ponen en cero, luego de haber estado todos en nivel alto, se memorizará una interrupción por teclado.

La sensibilidad al disparo de la interrupción del teclado que se haya programado en el respectivo *bit* de control (MODEK) debe te-

Figura 23.6 *Diagrama en bloques del módulo de interrupciones para teclado KBI*



nerse en cuenta para evitar perder algunas solicitudes de interrupción, ya que cuando la interrupción solo obedece al flanco de bajada, ésta puede ser bloqueada por cualquiera de las otras entradas de interrupción que permanezcan en cero. Esto no sucede cuando se ha seleccionado el disparo por flanco y nivel bajo, pues bajo esta condición la interrupción será reconocida tan pronto como cualesquiera de sus pines se coloque en cero.

Registro de estado y control del módulo KBI (KBSCR)

El registro de estado y control de las interrupciones para teclado llamado KBSCR, se usa para habilitar el reconocimiento y el enmascaramiento de tales interrupciones, y para configurar el tipo de sensibilidad al disparo. También uno de sus *bits* se usa como señalizador de la solicitud de interrupciones.

En la **figura 23.7** se muestra el registro KBSCR, y a continuación se describen las funciones de los *bits* que hacen parte del módulo KBI.

KEYF: Es el *bit* bandera que señaliza el estado de solicitud de una interrupción por teclado pendiente en el puerto A. Es de sólo lectura, y el *reset* lo borra.

Con **1** lógico: el *bit* indica que hay una interrupción por teclado pendiente.
Con **0** lógico: el *bit* indica que no hay una interrupción por teclado pendiente.

ACKK: es el *bit* de reconocimiento de la interrupción por teclado y es de sólo escritura.

Figura 23.7 *Registro de estado y control del módulo KBI (KBSCR)*

Al escribir un **1** lógico en este *bit* se borran todas las solicitudes de interrupción por teclado pendientes en el puerto A, y por medio del ciclo de *reset* es puesto en cero lógico.

IMASKK: es el *bit* de enmascaramiento de todas las interrupciones del módulo KBI, puede ser escrito o leído, y por medio del ciclo de *reset* será borrado.

Con **1** lógico: se enmascaran o inhabilitan todas las interrupciones del módulo KBI.
Con **0** lógico: se habilitan todas las interrupciones para teclado del KBI.

MODEK: es el *bit* que controla la sensibilidad del disparo de las interrupciones del KBI en los pines del puerto A, por medio del ciclo de *reset* se coloca en cero, y puede ser de lectura y escritura.

Con **1** lógico: se activa la sensibilidad de las interrupciones por flanco de bajada y nivel de voltaje bajo.
Con **0** lógico: se sensibiliza la solicitud de interrupciones por medio del flanco de bajada solamente.

Registro habilitador de las interrupciones del módulo KBI (KBIER)

Este registro se usa para habilitar o inhabilitar de manera individual cada una de las interrupciones en los pines del puerto A, que forman parte del módulo KBI. En la **figura 23.8** se muestra el registro de habilitación KBIER y enseguida se describen las funciones de los *bits* que hacen parte del módulo KBI.

KBIE6 a KBIE0: con cada uno de estos *bits* se pueden habilitar o no los pines del puerto A, que se asocian con cada una de las entradas de interrupción para teclado. Todos son de lectura y escritura, y por medio de un ciclo de *reset* serán borrados para inhabilitar las entradas KBI, y habilitar el registro del puerto A.

Con **1** lógico: cada uno de los *bits* de control KBIx habilitan el correspondiente pin

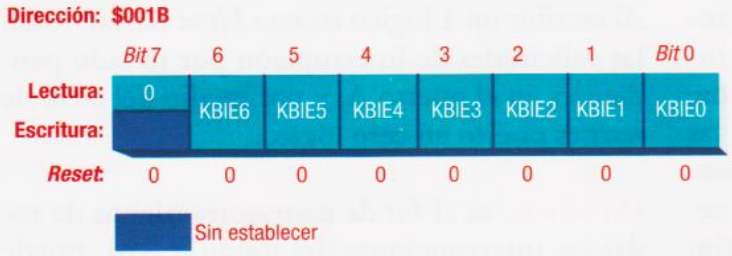


Figura 23.8 *Registro habilitador de las interrupciones del módulo KBI (KBIER)*

en el puerto A como una entrada de interrupción para teclado.

Con **0** lógico: cada uno de los *bits* de control KBIx inhabilitan el correspondiente pin en el puerto A como una entrada de interrupción para teclado, y lo definen como un puerto de propósito general (puerto A).

Nota:

La programación de una entrada de interrupción para teclado, no evita que deba ser configurada como una entrada.

Cuando el programador pone en uno lógico cualesquiera de los *bits* de habilitación de la interrupción para teclado KBIEx, obliga a que el pin del puerto de la interrupción correspondiente sea configurado por la CPU08 como una entrada; sin embargo, es importante recordarle al usuario que para leer dicho pin no debe olvidar poner en cero lógico, por medio de *software*, el correspondiente *bit* del registro de dirección de datos del puerto A (DDRA).

PARA SABER MÁS:

Inicialización de las entradas del módulo KBI

Cuando se habilita un pin de interrupción para teclado, le toma algo de tiempo alcanzar el uno lógico en virtud de la conexión de su resistencia interna de *pull-up*, y durante este breve periodo de tiempo, la CPU08 puede recibir falsas señales de interrupción. Para evitar que se produzcan interrupciones falsas durante la inicialización del módulo KBI, los interruptores de los registros de estado y control KBSCR y KBIER se deben activar de acuerdo con los siguientes pasos:

PASO 1. Use el *bit* IMASKK para enmascarar todas las interrupciones de teclado, colóquelo en uno lógico.

PASO 2. Use los *bits* adecuados KBIEx para habilitar de manera independiente los pines de las entradas de interrupción elegidas para teclado.

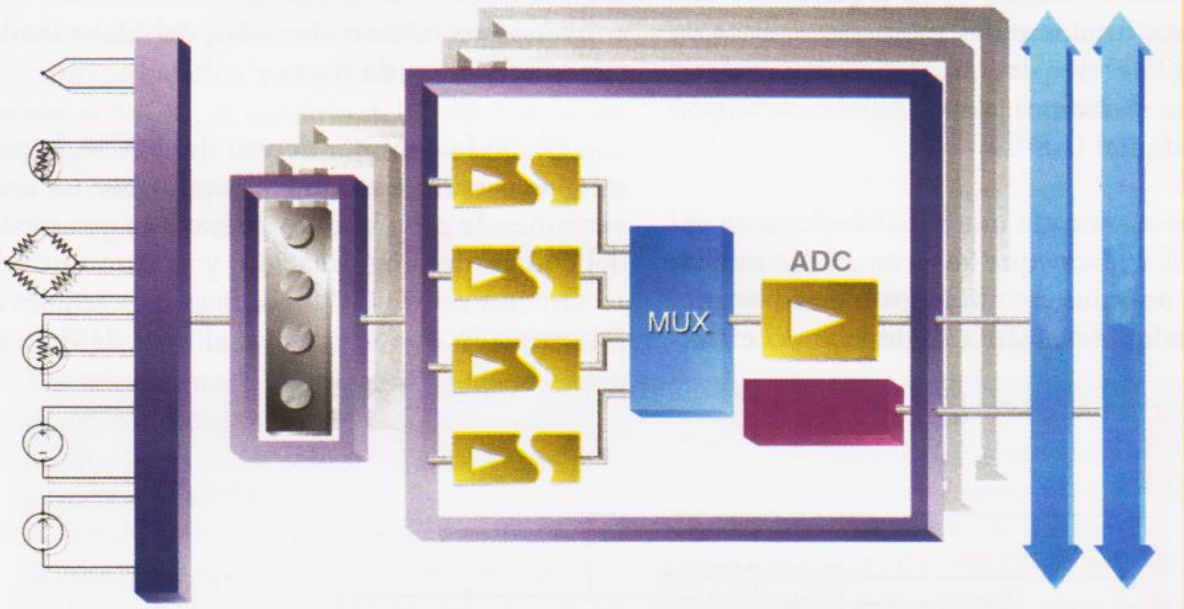
PASO 3. Borre cualquier solicitud de interrupción falsa y pendiente por medio del *bit* ACKK, al colocarlo en uno lógico

PASO 4. Borre el *bit* IMASKK para habilitar todas las interrupciones por teclado que hayan sido elegidas.

Una vez habilitadas las interrupciones, aquella que se presente en un pin sensible al flanco de bajada solamente será reconocida de inmediato; pero, cuando llege a una entrada sensible al flanco y al nivel bajo, el reconocimiento de la interrupción se hará luego de un tiempo de retraso que será función del tipo de carga externa allí presente.

Otra forma de evitar interrupciones falsas, consiste en configurar los pines del teclado como salidas, por medio de los *bits* adecuados del registro de dirección de datos del puerto A (DDRA), antes de habilitar las entradas de interrupción para el teclado por medio de los *bits* KBIEx.

LECCIÓN 24



EL MÓDULO CONVERTIDOR A/D (ADC) Y EL SIM DEL 68HC08

La capacidad del microcontrolador para recibir y procesar señales análogas, que pueden asumir cualquier valor en el tiempo a través de los ADC, ha sido muy importante porque amplió la gama de usos de estos dispositivos dentro de la industria y la investigación, permitiéndoles introducirse con gran fuerza en el delicado campo de la instrumentación electrónica.

El SIM es un módulo que en los microcontroladores de Motorola coordina y vigila todas las actividades que deben llevarse a cabo desde la CPU, así como el ingreso y retorno adecuado de los modos de operación especiales.

El módulo de conversión análoga a digital ADC

Para todos aquellos usos del microcontrolador, en el campo de las medidas, la supervisión, y la instrumentación de variables físicas con comportamiento analógico (señales que pueden asumir infinitos valores en el tiempo), que son muy frecuentes, los MCU de la familia HC08 de Motorola han incluido dentro de su arquitectura una herramienta fundamental para convertirlas en señales digitales (señales que sólo asumen valores discretos en el tiempo): el módulo de conversión análoga a digital (ADC).

La conversión de una señal análoga en digital es un trabajo que se debe hacer previamente, si se quiere recibir y procesar por medio de cualquier sistema digital como el microcontrolador. La tarea de conversión A/D se lleva a cabo a través de diversas técnicas, y cuenta con una amplia gama de circuitos integrados AD de varios fabricantes que la soportan; sin embargo, con el módulo ADC dentro del MCU se obtienen ventajas propias de la tecnología de integración, como son la reducción de espacio, el consumo de potencia y el costo; además, se minimizan los riesgos de efectuar y leer conversiones falsas o alteradas, debidas a inadecuados sistemas de tierra y cableado.

El módulo ADC, dentro del MCU, le permite al usuario un control total sobre las conversiones de cada uno de sus canales por medio del código de programación, y se comunica directamente con la CPU por medio de señales de interrupción que se generan al final de cada ci-

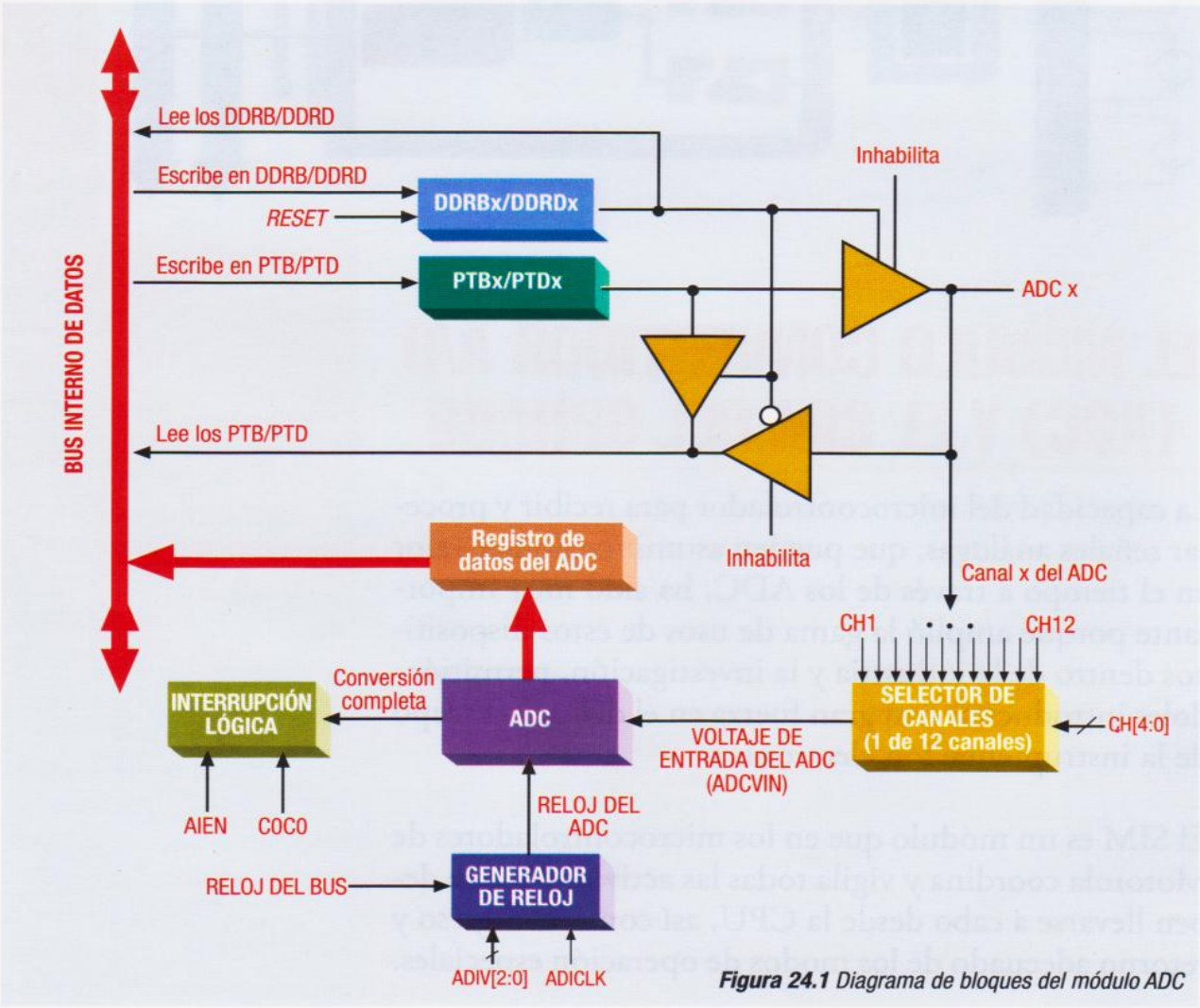


Figura 24.1 Diagrama de bloques del módulo ADC

clo de conversión; de esta manera, la CPU no resulta desviada de su rutina principal durante el tiempo de conversión de cada canal. Las características de este módulo son:

- Un convertidor A/D con doce canales de entrada de 8 *bits*, multiplexados.
- Técnica de conversión análoga a digital por aproximaciones sucesivas.
- Resolución de 8 *bits* por canal.
- Dos modos de conversión, continua o por ciclos.
- Dos indicadores de fin de conversión, por señalizador o por interrupción.
- Selector de la fuente del reloj de muestreo del ADC.

Descripción del módulo ADC

El módulo ADC se muestra en la **figura 24.1**, allí se ve el diagrama del circuito electrónico de este módulo representado en sus principales bloques.

Todos los canales de entrada análogos capacitados para muestrear señales externas, se encuentran disponibles a través de los terminales del MCU que corresponden con los puertos de propósito general PTB0 a PTB7, y PTD0 a PTD3; y por medio del registro de estado y control del ADC (ADSCR) se definen cuales de los doce canales estarán activos y se comportarán como entradas análogas.

Para resumir el comportamiento del módulo ADC, se puede anotar, que, por medio de un multiplexor análogo se elige uno de los doce canales A/D como la fuente de voltaje de entrada (ADCVIN) del único convertidor análogo a digital disponible en el módulo, y luego, este voltaje análogo, por medio de aproximaciones sucesivas, se convierte en una magnitud discreta representada en 8 *bits* en el registro de datos del ADC. Al finalizar un ciclo de conversión el módulo ADC graba el resultado con resolución de 8 *bits* en el registro de datos del ADC, llamado ADR; y pone en uno lógico su correspondiente *bit* bandera COCO, o bien, genera una solicitud de interrupción.

Voltaje de conversión ADCVIN

Los límites para el voltaje de conversión siempre están definidos por la magnitud de la fuente de alimentación del microcontrolador; de esta forma, para un voltaje de entrada igual o mayor a V_{DD}, el ADC entregará como resultado de la conversión un dato igual a $FF, mientras que para VSS o menos, el dato será $00. Todos los otros valores del voltaje de entrada que se encuentren entre V_{SS} y V_{DD}, obtendrán una conversión lineal y podrán ser representados por una línea escalonada con pendiente constante, y dividida en 256 valores discretos ($00 a $FF).

Nota:
Para evitar daños permanentes en los puertos que se comparten con los canales A/D, la magnitud del voltaje de entrada análogo ADCVIN nunca debe estar por encima ni por debajo de los umbrales de alimentación V_{DD} y V_{SS}, respectivamente.

Tiempo de conversión

Para completar un ciclo de conversión, el módulo ADC usa 16 estados de su reloj interno, y se inicia con el primer flanco creciente de dicho reloj. Por medio de la siguiente ecuación se puede calcular el tiempo que toma una conversión:

$$\text{Tiempo de conversión} = \frac{\text{16 períodos del reloj del ADC}}{\text{Frecuencia del reloj del ADC}}$$

Por ejemplo, para una frecuencia de reloj del ADC definida en 1MHz, el tiempo de un ciclo de conversión completo será de16s y la máxima frecuencia de muestreo será igual a 62,5kHz.

Modo de conversión continua

Una vez se activa el modo de conversión continua, el módulo ADC efectúa ciclos consecutivos de conversión A/D sobre el canal de entrada seleccionado, y sobrescribe y actualiza el contenido del registro de datos ADR al concluir cada tiempo de conversión. Para interrumpir el proceso de conversión continua, se requiere borrar

el *bit* de control ADCO que hace parte del registro de estado y control del ADC.

En este modo, el *bit* de conversión completa COCO se coloca en uno después de cada conversión exitosa; y el usuario podrá borrarlo por *software* escribiendo directamente sobre el registro ADSCR, o, leyendo el registro de datos ADR.

Las interrupciones del ADC

Al colocar un uno lógico en el *bit* habilitador de la interrupción del convertidor A/D, llamado AIEN, el módulo es capaz de generar una solicitud de interrupción a la CPU08 al cabo de cada ciclo de conversión. Cuando la interrupción del ADC está activa, el *bit* COCO no se usa como bandera del fin de conversión, sino que se usa para disparar la solicitud de interrupción cuando su estado es cero lógico.

Registros de configuración del módulo ADC

Los registros especiales del microcontrolador que se usan para definir los parámetros del módulo convertidor ADC, son tres: el registro de estado y control ADSCR, el registro de datos ADR, y el registro de configuración del reloj ADICLK. En los siguientes párrafos se hace la descripción de cada uno de los *bits* que conforman estos registros.

1. Registro de estado y control del ADC (ADSCR)

En la **figura 24.2** se muestra el registro ADSCR que nos permite configurar los canales de conversión activos, habilitar la interrupción, seleccionar su modo de trabajo, y conocer el estado de la conversión.

COCO: es el *bit* bandera que señaliza cuando una conversión se completa, es de sólo lectura; y puede ser automáticamente borrado al escribir directamente en el registro ADSCR, o, al leer el registro ADR. El *reset* lo borra.

Figura 24.2 *Registro de estado y control del ADC (ADSCR)*

Cuando AIEN = 0:
Con **1** lógico, indica conversión completa
Con **0** lógico, indica conversión incompleta

CH4	CH3	CH2	CH1	CH0	Canal ADC	Selector de entradas
0	0	0	0	0	ADC0	PTB0
0	0	0	0	1	ADC1	PTB1
0	0	0	1	0	ADC2	PTB2
0	0	0	1	1	ADC3	PTB3
0	0	1	0	0	ADC4	PTB4
0	0	1	0	1	ADC5	PTB5
0	0	1	1	0	ADC6	PTB6
0	0	1	1	1	ADC7	PTB7
0	1	0	0	0	ADC8	PTD3
0	1	0	0	1	ADC9	PTD2
0	1	0	1	0	ADC10	PTD1
0	1	0	1	1	ADC11	PTD0
0	1	1	0	0		
:	:	:	:	:	-	No usado
1	1	0	1	0		
1	1	0	1	1	-	Reservado
1	1	1	0	0	-	No usado
1	1	1	0	1		V_{DDA} (Nota 1)
1	1	1	1	0		V_{SSA} (Nota 1)
1	1	1	1	1		Apagado del ADC

Nota 1:
Los voltajes internos de referencia se usan para verificar la operación del convertidor ADC.

Tabla 24.1 *Tabla de selección de canales del módulo ADC*

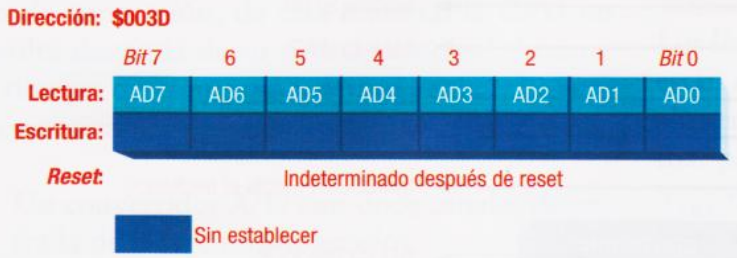


Figura 24.3 *Registro de datos del ADC (ADR)*

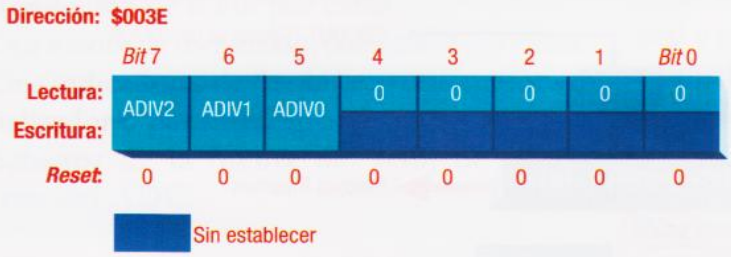


Figura 24.4 *Registro de configuración del reloj del ADC (ADICLK)*

AIEN: es el *bit* de control de la interrupción del ADC, es de escritura, y cuando se coloca, el módulo ADC genera una interrupción al completar la conversión. Las condiciones que borran este *bit* son iguales a las del COCO.

Con **1** lógico, se habilita la interrupción.
Con **0** lógico, se inhabilita la interrupción.

ADCO: es el *bit* selector del modo de conversión continua o por ciclos del módulo ADC, es de escritura, y nos permite elegir entre el modo de conversión continua y la conversión de un sólo ciclo.

Con **1** lógico, se activa el modo de conversión continua.
Con **0** lógico, se inactiva la conversión continua y se activa la conversión de un sólo ciclo.

ADCH[4:0]: son los *bits* del selector de canales ADC, con los *bits* CH0 a CH3 se elige uno de los doce canales disponibles, y con el CH4 se les asignan algunas funciones especiales. La **tabla 24.1** resume las condiciones de estos *bits*.

2. Registro de datos del ADC (ADR)

En la **figura 24.3** se muestra el registro de un *byte* que contiene en todo momento el resultado de la última conversión hecha por el módulo ADC, llamado ADR; es de sólo lectura, y su contenido es inmediatamente actualizado cada vez que una conversión termina. Después de un *reset* su estado puede ser cualquiera.

3. Registro de configuración del reloj del ADC (ADICLK)

En la **figura 24.4** se muestra el registro ADICLK que nos permite seleccionar la frecuencia del reloj interno para el muestreo de las señales con el ADC, por medio de los *bits* ADIV0 a ADIV2 se puede seleccionar el divisor usado por este módulo para generar el reloj interno del ADC. La **tabla 24.2** muestra las opciones para los *bits* ADIVx.

El módulo del sistema de integración SIM

El módulo del sistema de integración SIM se representa en bloques en la **figura 24.5**; el SIM es el controlador del estado del sistema y se encarga de coordinar todas las actividades entre la CPU y el sistema de buses. El SIM es responsable de:

ADIV2	ADIV1	ADIV0	Divisor del reloj del ADC
0	0	0	Reloj de entrada ADC÷1
0	0	1	Reloj de entrada ADC÷2
0	1	0	Reloj de entrada ADC÷4
0	1	1	Reloj de entrada ADC÷8
1	X	X	Reloj de entrada ADC÷16

Tabla 24.2 *Tabla de preescaladores del reloj del ADC (ADIVx)*

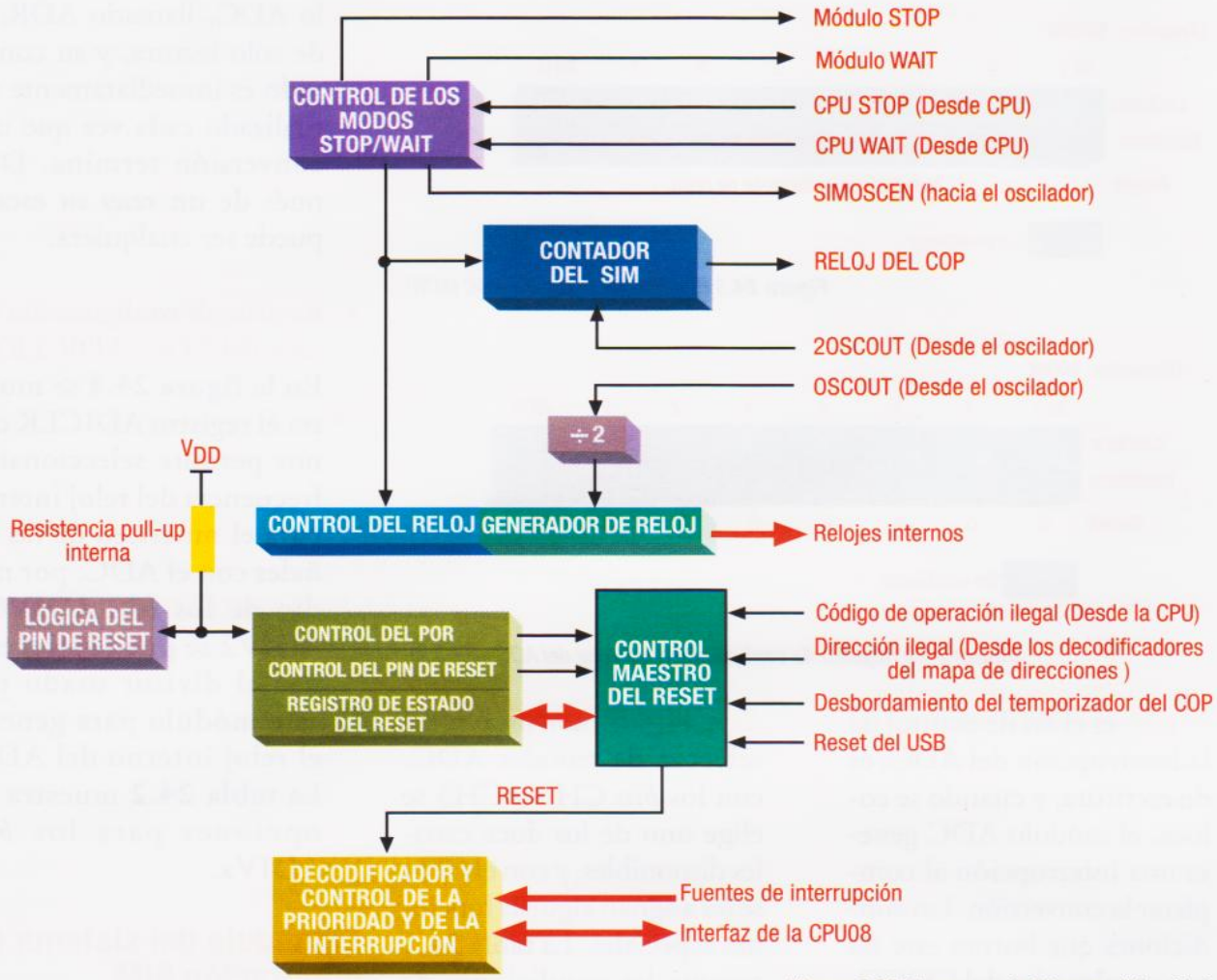


Figura 24.5 *Diagrama de bloques del módulo SIM*

- Controlar la CPU08 y los periféricos.
- Controlar la entrada y salida de los modos especiales de funcionamiento: STOP, WAIT, RESET, y BREAK.
- Controlar el reloj interno del bus.
- Controlar el *reset* maestro, el de la conexión (POR), y el desbordamiento del COP.
- Controlar el procesamiento de todas las interrupciones externas e internas.
- Controlar la arquitectura modular expansible de las fuentes de interrupción.

Fuentes internas de interrupción del reset

Para completar nuestro estudio de la interrupción principal del MCU, el *reset*, en esta ocasión se hará énfasis sólo en las cinco fuentes internas que lo originan, ya que su estructura y las fuentes externas fueron ampliamente tratadas en otra ocasión.

Figura 24.6 *Fuentes de interrupción internas de la CPU08*

Todas las interrupciones internas de *reset* que afectan a la CPU08 se muestran en la **figura 24.6**, estas son: dirección ilegal, código de operación ilegal, desbordamiento del temporizador del COP, el *reset* a la conexión (POR), y el inhibidor por bajo voltaje (LVI). Todas ellas fuerzan el pin de entrada externa del RST a un nivel bajo durante 32 periodos del reloj 2OSCOUT, para permitir la inicialización de los periféricos externos conectados alrededor del microcontrolador. En la **figura 24.7** se muestra el diagrama de tiempos de la vectorización de una interrupción de este tipo.

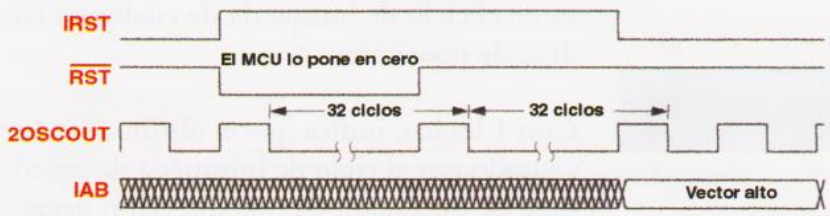


Figura 24.7 *Diagrama de tiempos de un reset interno*

El reset a la conexión de la alimentación POR

Cuando se conecta por primera vez la alimentación al MCU, el módulo del POR (*power-on reset*) genera un pulso para indicar a la CPU que esto ha sucedido, y se da inicio a la secuencia de acciones propias del POR, que son:

1. Se reconoce la fuente de *reset* interna.
2. El SIM habilita la salida de reloj 2OSCOUT.
3. Los relojes internos de la CPU y los módulos se detienen durante 4.096 periodos del reloj 2OSCOUT, mientras que el módulo del oscilador se estabiliza.
4. El pin de entrada $\overline{RST}$, se coloca en nivel bajo durante el tiempo de estabilización del oscilador.
5. El *bit* POR del registro de estado del *reset* del SIM (RSR), se graba con uno lógico, mientras que todos los otros *bits* de este registro son borrados.

El *reset* por código de operación ilegal

El SIM decodifica las señales que se originan en la CPU08 para detectar instrucciones ilegales, cualquier instrucción ilegal pone en uno lógico el *bit* ILOP del registro RSR, y genera una solicitud de interrupción por *reset* interno.

El reset por dirección ilegal

Cuando el SIM se da cuenta que la CPU ha iniciado el ciclo de búsqueda de un código de operación desde una dirección ilegal (fuera del área del mapa de memoria definido para la CPU08), se genera un *reset* por dirección ilegal, y el SIM coloca en uno lógico el *bit* de estado ILAD del registro RSR, inmediatamente antes de reiniciar el MCU.

El reset de los módulos COP y LVI

De estos dos módulos, ya estudiados en detalle en la lección anterior, solo se dirá que ambos generan un *reset* interno cada que se desborda el contador del COP, o cuando V_{DD} cae por debajo del umbral (VTRIP) definido en el módulo LVI. En cualquier caso, el proceso de la interrupción sólo se iniciará a los 64 periodos del reloj 2OSCOUT, y después la estabilidad del oscilador.

Los registros de estado del módulo SIM

Los registros del SIM son tres y están localizados en la memoria entre las posiciones $FE00 y $FE03, por medio de sus *bits* el programador puede conocer el estado de la CPU08 en todo momento.

1. El registro de estado del modo BREAK (BSR).
   En la **figura 24.8** se muestra el registro BSR, este registro de estado posee un *bit* bandera que indica cuando la salida de los modos STOP y WAIT se ocasiona por un *break*, y se le conoce como SBSW; para borrarlo se requiere de un *reset*, o, escribir un cero en él.

   Con **1** lógico, indica que la CPU salió de los modos STOP o WAIT debido a una interrupción *break*.
   Con **0** lógico, indica que la CPU no salió de los modos STOP o WAIT debido a una interrupción *break*.

2. El registro de estado del reset (RSR).
   En la figura 24.9 se muestra el registro RSR, este registro posee los *bits* bandera que infor-

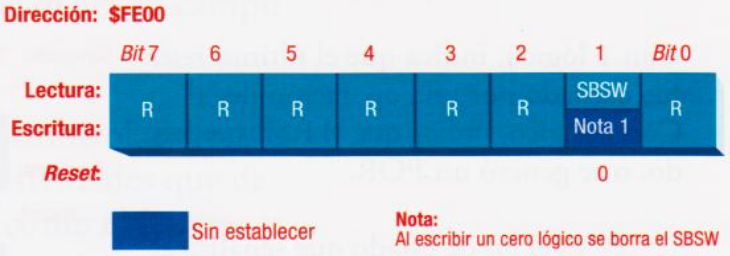


Figura 24.8 *Registro de estado del modo break (BSR)*

Dirección: $FE01

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura:	POR	PIN	COP	ILOP	ILAD	MODRST	LVI	0
Escritura:								
POR:	1	0	0	0	0	0	0	0

Sin establecer

Figura 24.9 *Registro de estado del reset (RSR)*

man al usuario sobre la procedencia del último *reset* ejecutado. Para borrar todas sus banderas automáticamente (excepto la del POR), basta leerlo.

POR: es el *bit* de estado que señaliza al *reset* a la conexión (*Power-On Reset*).

Con **1** lógico, indica que el último *reset* fue causado por el circuito POR.
Con **0** lógico, indica que el registro RSR fue leído.

PIN: es el *bit* de estado que señaliza al *reset* externo RST.

Con **1** lógico, indica que el último *reset* fue causado por el pin de entrada externo $\overline{\text{RST}}$.
Con **0** lógico, indica que el RSR fue leído, o se generó un POR.

COP: es el *bit* de estado que señaliza el desbordamiento del temporizador de vigilancia del COP.

Con **1** lógico, indica que el último *reset* fue causado por el COP.
Con **0** lógico, indica que el RSR fue leído, o se generó un POR.

ILOP: es el *bit* de estado que señaliza al *reset* causado por una operación ilegal.

Con **1** lógico, indica que el último *reset* fue causado por una operación ilegal.
Con **0** lógico, indica que el RSR fue leído, o se generó un POR.

ILAD: es el *bit* de estado que señaliza al *reset* debido a una dirección ilegal durante el ciclo de búsqueda de cualquier código de operación.

Con **1** lógico, indica que el último *reset* fue causado por el ciclo de búsqueda de un código de operación en una dirección ilegal.
Con **0** lógico, indica que el RSR fue leído, o se generó un POR.

MODRST: es el *bit* de estado que señaliza el *reset* del modo monitor.

Con **1** lógico, indica que el último *reset* fue causado por la entrada al modo monitor.
Con **0** lógico, indica que el RSR fue leído, o se generó un POR.

LVI: es el *bit* de estado que señaliza el *reset* causado por el módulo LVI.

Con **1** lógico, indica que el último *reset* fue causado desde el circuito LVI.
Con **0** lógico, indica que el RSR fue leído, o se generó un POR.

3. El registro de control de la bandera del break (BFCR).

En la **figura 24.10** se muestra el registro BFCR, este registro contiene un *bit* de lectura y escritura que habilita al programador desde el *software* para borrar los *bits* de estado mientras que la CPU está en estado de *break*, se llama BCFE. Con el *reset* se pone en cero.

Con **1** lógico, se habilita el borrado de todos los *bits* de estado durante el *break*.
Con **0** lógico, se inhabilita el borrado durante el estado de *break*.

Dirección: $FE03

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura: / Escritura:	BCFE	R	R	R	R	R	R	R
Reset:	0							

R Reservado

Figura 24.10 *Registro de control de la bandera del break (BFCR)*

LECCIÓN 25

ELMÓDULO DE LA INTERFAZ DE TEMPORIZACIÓN (TIM) DEL 68HC08

La cuenta del tiempo es la tarea más común dentro de las aplicaciones de los microcontroladores en cualesquiera de sus campos, y, por ello, el primer circuito periférico interno de propósito especial que se incorporó a la arquitectura de estos dispositivos fue el módulo temporizador.

Gracias a los módulos temporizadores, el programador puede contar cualquier periodo de tiempo usando como base el ciclo del reloj interno del microcontrolador, y, sin ocupar a la CPU.

La temporización de los eventos, por medio de los lazos de *software*, fue la primera forma de contar el tiempo en los microcontroladores; ésta es una tarea simple, pero, no obstante, puede ocupar durante mucho tiempo a la CPU e impedir que ella desarrolle otras actividades reduciendo considerablemente la eficiencia del dispositivo. Este tipo de temporización está además limitada por el tamaño y el número de registros de propósito general disponibles, y, por lo tanto, sólo se recomienda para contar pequeños periodos de tiempo.

Gracias a la integración de circuitos temporizadores como dispositivos periféricos dentro de la arquitectura de los microcontroladores, se le ha facilitado esta tarea al programador, y se ha aumentado la eficiencia de los MCU considerablemente, permitiéndonos, además de contar periodos largos de tiempo, llevar a cabo otras funciones que se basan en la cuenta del tiempo, tales como la transmisión y recepción de datos, la vigilancia de procesos, y el manejo y control de otros dispositivos por medio de las técnicas clásicas de modulación de pulsos.

El TIM del 68HC08

En la **figura 25.1** se muestra el diagrama de bloques del TIM (*timer interface module*). En esta lección estudiaremos la segunda versión del TIM de los microcontroladores Motorola de la familia HC08 (TIM2, versión B); éste es el módulo de la interfaz de temporización, y está compuesto por

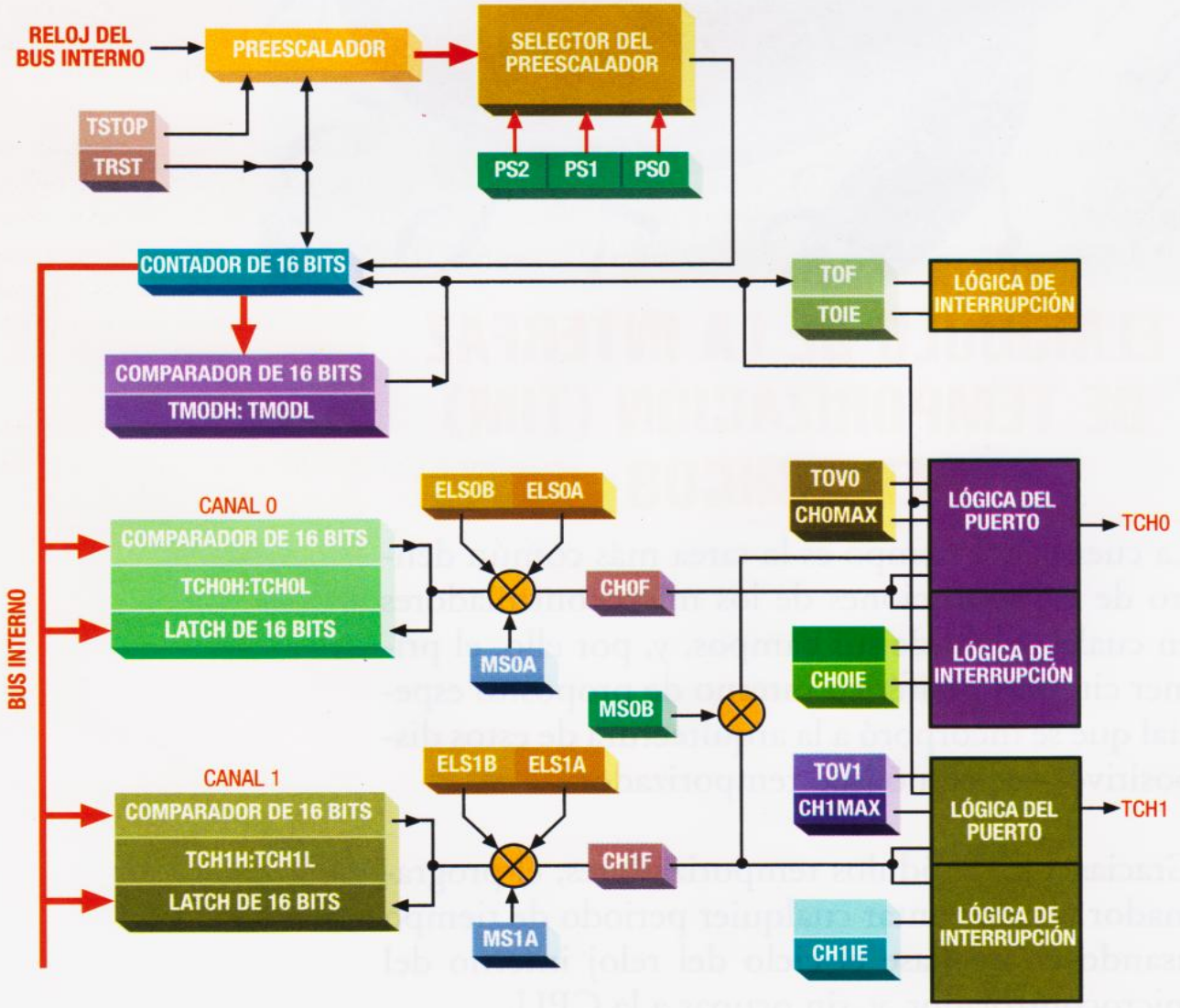


Figura 25.1 *Diagrama de bloques del TIM del 68HC08*

un temporizador de dos canales con entrada de captura, salida de comparación, y funciones de modulación por ancho de pulsos (PWM). Las principales características que posee este módulo son:

- Dos canales independientes con entrada de captura y salida de comparación
- Generador de salida PWM con *buffer* o sin *buffer*
- Preescalador programable para el reloj del TIM
- Contador de carrera-libre, o en módulo contador ascendente
- *Bits* de paro y *reset* del contador del TIM
- Arquitectura modular (expansible hasta ocho canales)

Funciones del TIM

El TIM comparte sus dos canales de entrada y salida con los dos puertos I/O de propósito general, PTD4 y PTD5; el principal componente del TIM es el contador de 16 *bits*, de él proviene la referencia para las funciones de captura y comparación. Usando los registros de 8 *bits* TMODH y TMODL el programador puede controlar el valor del módulo contador del TIM, y puede leerlo en cualquier momento sin afectar la secuencia de la cuenta.

En la estructura del TIM que se muestra en la figura 25.1 se destacan las siguientes partes:

- **El preescalador del contador del TIM**
  El bloque preescalador usa los *bits* PS[2:0], localizados en el registro de estado y control del TIM (TSC), para seleccionar la fuente del reloj, y la salida de reloj puede ser elegida entre siete opciones de frecuencia que se obtienen al dividir el reloj interno del bus.

- **La entrada de captura**
  Por medio de las funciones de este bloque el TIM puede capturar el tiempo durante el cual ocurre algún evento externo. Cuando un flanco activo programado se detecta por el pin de un canal configurado como una entrada de captura, el TIM guarda el contenido del contador en los registros del canal correspondiente, TCHxH y TCHxL, y, adicionalmente, puede generar una solicitud de interrupción en la CPU08.

- **La salida de comparación**
  Usando las funciones definidas para el circuito de comparación del TIM, el MCU puede generar pulsos periódicos, con polaridad, duración, y frecuencia programables. Cuando el contador iguala el valor guardado en los registros de un canal configurado como salida de comparación, el TIM puede poner, borrar, o cambiar el estado del pin asociado a dicho canal; y, adicionalmente, puede generar una solicitud de interrupción en la CPU08.

Cualesquiera de los dos canales programados como salidas de comparación pueden generar pulsos directamente sin la intervención de un *buffer*, y esto quiere decir, que para cambiar el valor de comparación de la salida, que está guardado en los registros del canal del TIM correspondientes, solo se requiere sobreescribir el nuevo valor en tales registros.

Los canales cero y uno (CH0 y CH1), también pueden enlazarse para configurar una salida de comparación con *buffer* disponible a través de la salida del canal cero (pin PTD4/TCH0).

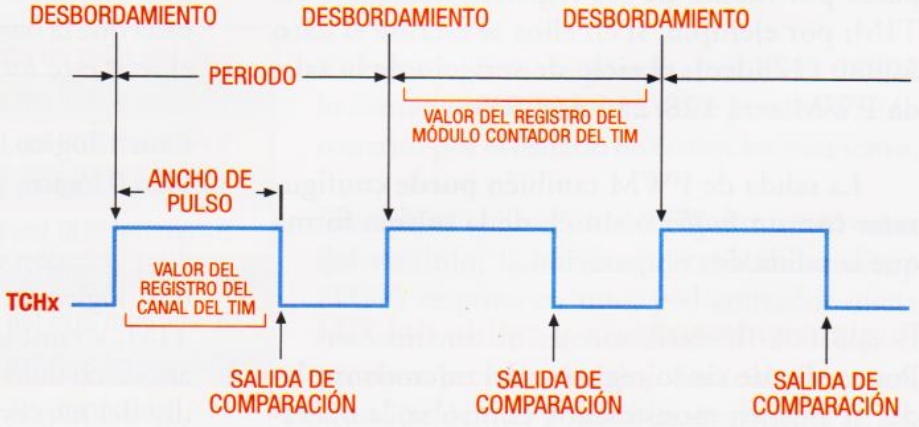


Figura 25.2 *Señal PWM generada por el TIM del 68HC08*

El modulador PWM

La característica que le permite al TIM cambiar el estado de la salida de comparación de cualquier canal al desbordar el contador (*toggle-on- overflow*), se usa para generar una señal de salida modulada por ancho de pulsos (PWM). Con el valor de los registros del módulo contador se determina el periodo de la señal PWM, y el estado del pin del canal cambia cuando el contador alcanza el valor del registro del módulo contador.

En la **figura 25.2** se muestra una señal PWM, el periodo, y el ancho del pulso de modulación; allí podemos ver que el periodo de la señal PWM está determinado por dos desbordamientos sucesivos del registro del módulo contador, mientras que el valor guardado en los registros del canal del TIM define el ancho del pulso de la señal modulada a través de la salida de comparación. El tiempo que transcurre entre un desbordamiento y una salida de comparación, es el ancho del pulso modulado.

Por medio de los valores de los registros del módulo contador y de la salida del preescalador se define la frecuencia de la señal de salida PWM, de forma tal que, si los *bits* del preescalador se ponen en 000, y al registro del módulo contador se le escribe $00FF, se generará una salida PWM con un periodo igual a 256 veces el estado del reloj del bus interno. El ancho del pulso también puede variarse en 256 pasos por medio de los registros del canal del TIM; por ejemplo, si en ellos se escribe el dato $0080 (128dec), el ciclo de servicio de la salida PWM será 128/256 ó el 50%.

La salida de PWM también puede configurarse con un *buffer* o sin él, de la misma forma que la salida de comparación.

Registros de entrada y salida del TIM

Por medio de cinco registros del microcontrolador se pueden monitorear y controlar, la operación y el estado del TIM; los registros son:

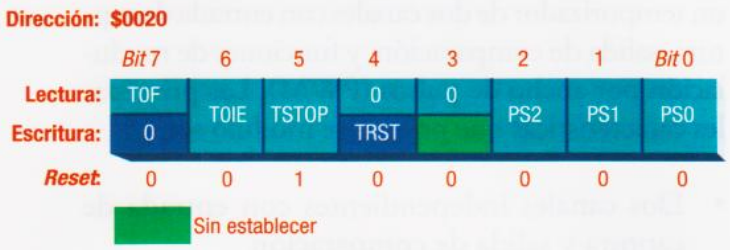


Figura 25.3 *Registro de estado y control del TIM (TSC)*

1. Registro de estado y control del TIM (TSC)

En la **figura 25.3** se muestra el contenido del registro TSC; por medio de este registro se pueden habilitar las interrupciones, consultar las banderas de estado, detener e inicializar el contador, y seleccionar la frecuencia del reloj.

A continuación se describen sus *bits*:

TOF: este *bit* de lectura y escritura es la bandera que se pone cuando el contador del TIM se inicializa en $0000, después de alcanzar el valor programado en los registros del módulo contador. Al leer el registro TSC se borra la bandera automáticamente, también, al escribir directamente en TOF un cero lógico, u, ocasionando un *reset*.

Con **1** lógico, indica que el contador del TIM ha alcanzado el valor del módulo.
Con **0** lógico, indica que el contador del TIM no ha alcanzado el valor del módulo.

TOIE: este *bit* de lectura y escritura nos permite activar la solicitud de las interrupciones por desbordamiento del contador del TIM, cada que la bandera TOF se pone en uno. Con el *reset* este *bit* se borra.

Con **1** lógico, se habilitan las interrupciones.
Con **0** lógico, se inhabilitan las interrupciones.

TSTOP: con este *bit* de lectura y escritura en uno lógico se puede detener el contador del TIM, y también permitir que el conteo se reanude cuando se pone en cero lógico. Por medio del *reset* este *bit* se pone en uno, y el programador siempre debe borrarlo por *software*.

Con **1** lógico, se detiene el contador del TIM.
Con **0** lógico, se activa el contador del TIM.

TRST: este *bit* de escritura solamente, se puede grabar con uno lógico para inicializar el contador y el preescalador del TIM, sin afectar a los demás registros. Cuando el *bit* TRST se pone, el contador inicia desde $0000, y tras la acción de *reset*, siempre será borrado.

Con **1** lógico, se borran el contador y el preescalador del TIM.
Con **0** lógico, no se produce ningún efecto sobre el TIM o la CPU08.

PS[2:0]: estos *bits* pueden ser leídos o escritos, y son los selectores del preescalador del reloj del contador del TIM, nos sirven para definir en la salida del circuito preescalador una de siete frecuencias configurables. Todos los *bits* son borrados por el *reset*.

En la **tabla 25.1** se resumen las opciones de selección disponibles para el preescalador del reloj del contador del TIM.

2. Registros del contador del TIM (TCNTH:TCNTL)

En la **figura 25.4** se muestran los contenidos de los dos registros de 8 *bits* que conforman el contador del TIM, TCNTH y TCNTL; éstos se ubican en las posiciones $0021 y $0022 del mapa de memoria y en ellos se guarda el *byte* alto y bajo del contador; ambos son registros de sólo lectura. Al leer por primera vez el valor del *byte* más significativo (TCNTH) automáticamente se guarda el contenido del de menor peso (TCNTL) en un *buffer* temporal, mientras que al poner el *bit* TRST en uno lógico, o, al generar un ciclo de *reset*, los dos registros del contador del TIM serán borrados.

PS2	PS1	PS0	Fuente de reloj del TIM
0	0	0	Reloj del BUS interno ÷ 1
0	0	1	Reloj del BUS interno ÷ 2
0	1	0	Reloj del BUS interno ÷ 4
0	1	1	Reloj del BUS interno ÷ 8
1	0	0	Reloj del BUS interno ÷ 16
1	0	1	Reloj del BUS interno ÷ 32
1	1	0	Reloj del BUS interno ÷ 64
1	1	1	No disponible

Tabla 25.1 *Tabla de selección del preescalador del reloj del TIM*

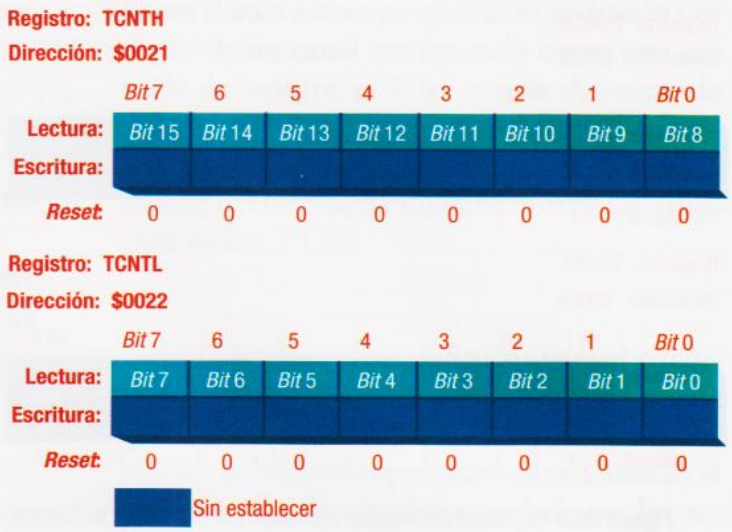


Figura 25.4 *Registros del contador del TIM (TCNTH:TCNTL)*

3. Registros del módulo contador del TIM (TMODH:TMODL)

En la **figura 25.5** se muestran los contenidos de los dos registros de 8 *bits* que hacen parte del módulo contador del TIM, TMODH y TMODL, los cuales se encuentran localizados en las posiciones $0023 y $0024 del área de memoria.

En estos registros se guarda el valor del módulo contador del TIM, y su valor puede ser leído o escrito por el usuario en cualquier momento.

Cuando el contador del TIM alcanza el valor del módulo, la bandera de desbordamiento (TOF) se pone en uno, y el contador inicia nuevamente su cuenta desde $0000 con el próximo periodo de reloj. En cualquier caso, la acción de *reset* pondrá en unos ($FFFF) los registros del módulo contador del TIM.

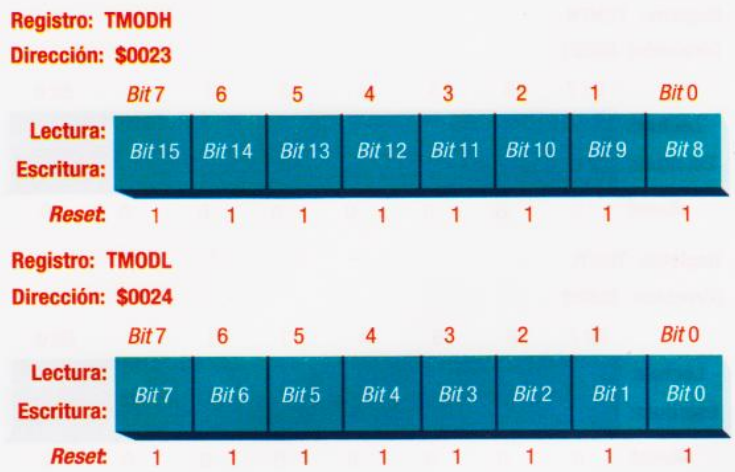


Figura 25.5 *Registros del módulo contador del TIM (TMODH:TMODL)*

4. Registros de estado y control de los canales del TIM (TSC0:TSC1)

En la **figura 25.6** se muestra la distribución de los registros TSC0 y TSC1, y la acción del *reset* sobre todos sus *bits*. A continuación se describen sus *bits*:

CHxF: este *bit* de lectura y escritura es la bandera de cada canal; cuando el canal está configurado como entrada de captura, el *bit* se pone en uno para indicar la detección de un flanco activo; mientras que cuando el canal está configurado como salida de comparación, la bandera se pone cuando el valor del registro contador del TIM se iguala con el valor del registro del respectivo canal.

Con **1** lógico, señala la detección de la entrada de captura o la salida de comparación en el respectivo canal.
Con **0** lógico, señala que no ha detectado la entrada de captura o la salida de comparación en el respectivo canal.

CHxIE: este *bit* de lectura y escritura nos permite habilitar la solicitud del servicio de interrupción de la CPU08 asociada con cada canal del TIM.

Con **1** lógico, se habilita la solicitud de interrupción del canal x.
Con **0** lógico, se inhabilita la solicitud de interrupción del canal x.

MSxB: es el *bit* selector de modo B, con este *bit* de lectura y escritura se selecciona el *buffer* en la salida de comparación, o en la del PWM. Este *bit* sólo está en el registro de estado y control del canal 0 (TSC0), y cuando el MS0B se pone en uno, se bloquean las funciones del registro TSC1 convirtiendo nuevamente al canal 1 en un puerto I/O de propósito general (PTD5).

Con **1** lógico, se habilita la salida con *buffer*
Con **0** lógico, se inhabilita la salida con *buffer*.

MSxA: es el *bit* selector de modo A, con este *bit* de lectura y escritura se selecciona la operación de la salida de comparación y PWM sin *buffer*, o la entrada de captura, o, el nivel de voltaje presente inicialmente en la salida.

Cuando ELSxB:A es diferente de 00:
Con 1 lógico, se habilita la salida de comparación o PWM sin *buffer*.
Con 0 lógico, se habilita como entrada de captura.

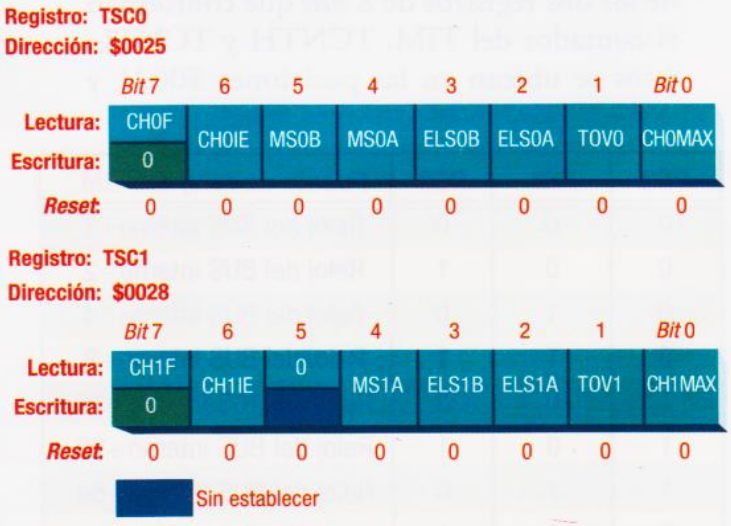


Figura 25.6 *Registros de estado y control de los canales del TIM (TSC0:TSC1)*

MSxB	MSxA	ELSxB	ELSxA	MODO	CONFIGURACIÓN
X	0	0	0	Salida estándar	Pin bajo el control del puerto; Inicialmente con la salida en nivel alto
X	1	0	0		Pin bajo el control del puerto; Inicialmente con la salida en nivel bajo
0	0	0	1	Entrada de captura	Captura en flanco creciente solamente
0	0	1	0		Salida en flanco decreciente solamente
0	0	1	1		Captura en flanco creciente o decreciente
0	1	0	1	Salida de comparación ó PWM	Cambia el estado de la salida bajo comparación
0	1	1	0		Borra la salida bajo comparación
0	1	1	1		Coloca la salida bajo comparación
1	X	0	1	Salidas de comparación ó PWM con buffer	Cambia el estado de la salida bajo comparación
1	X	1	0		Borra la salida bajo comparación
1	X	1	1		Coloca la salida bajo comparación

Tabla 25.2 *Tabla de selección del modo de operación, flanco y nivel del TIM*

Cuando ELSxB:A es igual a 00:
Con 1 lógico, se activa la salida inicial del canal x en nivel bajo.
Con 0 lógico, se activa la salida inicial del canal x en nivel alto.

ELSxB y ELSxA: son los *bits* selectores de flanco y nivel, con ellos se selecciona el flanco activo de la entrada de captura, o el nivel de la salida de comparación del canal correspondiente.

Cuando estos dos *bits* son borrados, el canal x se desconecta del puerto, y libera el pin del canal (TCHx) para que se comporte como un puerto I/O de propósito general.

En la **tabla 25.2** se muestran las opciones de configuración que estos *bits* nos ofrecen.

TOVx: este *bit* de lectura y escritura controla el comportamiento de la salida de comparación de cualquier canal cuando el contador del TIM se desborda.

Con **1** lógico, el pin de salida del canal x cambia de estado cuando el contador se desborda.
Con **0** lógico, el pin de salida del canal x no cambia de estado cuando el contador se desborda.

CHxMAX: este *bit* de control le permite al usuario determinar el máximo ciclo de servicio (igual al 100%, o equivalente al nivel VDD) para la señal de salida modulada por ancho de pulsos (PWM).

Cuando el *bit* TOVx está en cero, un uno lógico en CHxMAX obliga a que la salida PWM tenga un ciclo de servicio del 100%; en la **figura 25.7** se muestra el comportamiento de la salida PWM de cualquier canal de acuerdo con el estado de este *bit*, en ella se aprecia que, al poner a CHxMAX en cero lógico, la señal de salida PWM recupera su condición normal, y también que la salida del canal siempre responde a un ciclo después de haberlo borrado.

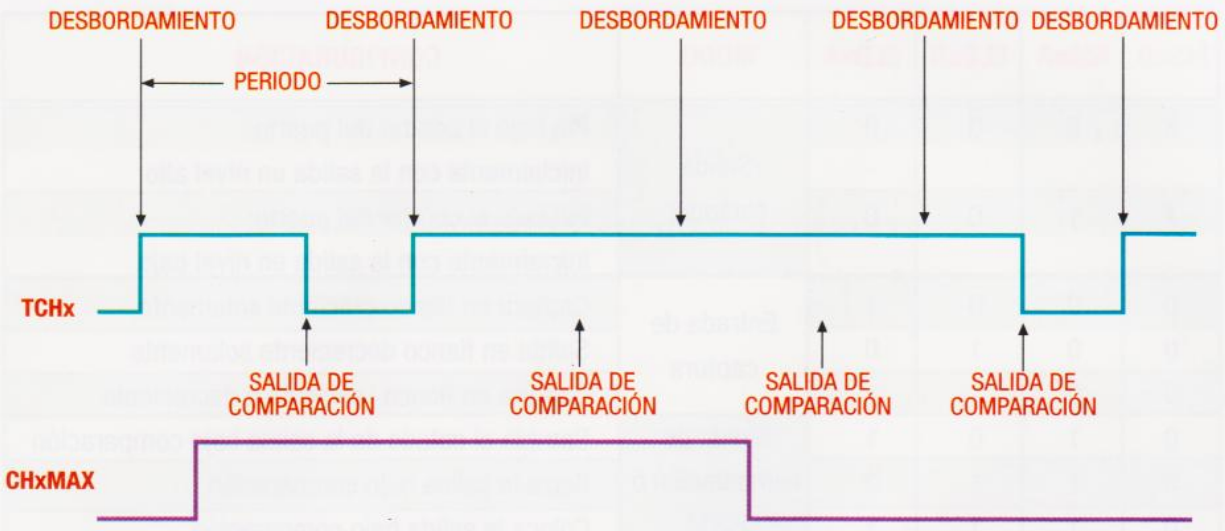


Figura 25.7 *Características y efectos del bit de control CHxMAX en la salida PWM*

5. Registros de los canales del TIM (TCH0H/L:TCH1H/L)

En la **figura 25.8** se muestra la distribución de los registros altos y bajos del los dos canales del TIM, TCH0H/L y TCH1H/L.

Estos registros de lectura y escritura están destinados a almacenar el valor capturado del contador del TIM cuando el canal x está funcionando como entrada de captura, o el valor comparado en la salida cuando funciona como salida de comparación.

En el modo de entrada de captura (con MSxB:MSxA = 00), la lectura del *byte* alto del registro del canal x (TCHxH) bloquea la entrada de captura hasta que la parte baja (TCHxL) sea leída; y, de forma similar, en el modo de salida de comparación (con MSxB:MSxA 00), al escribir el *byte* alto del registro del canal x (TCHxH) se bloquea la salida de comparación hasta que la parte baja (TCHxL) sea también escrita. En cualesquiera de los registros, la acción de un ciclo de *reset* siempre pondrá un dato indeterminado .

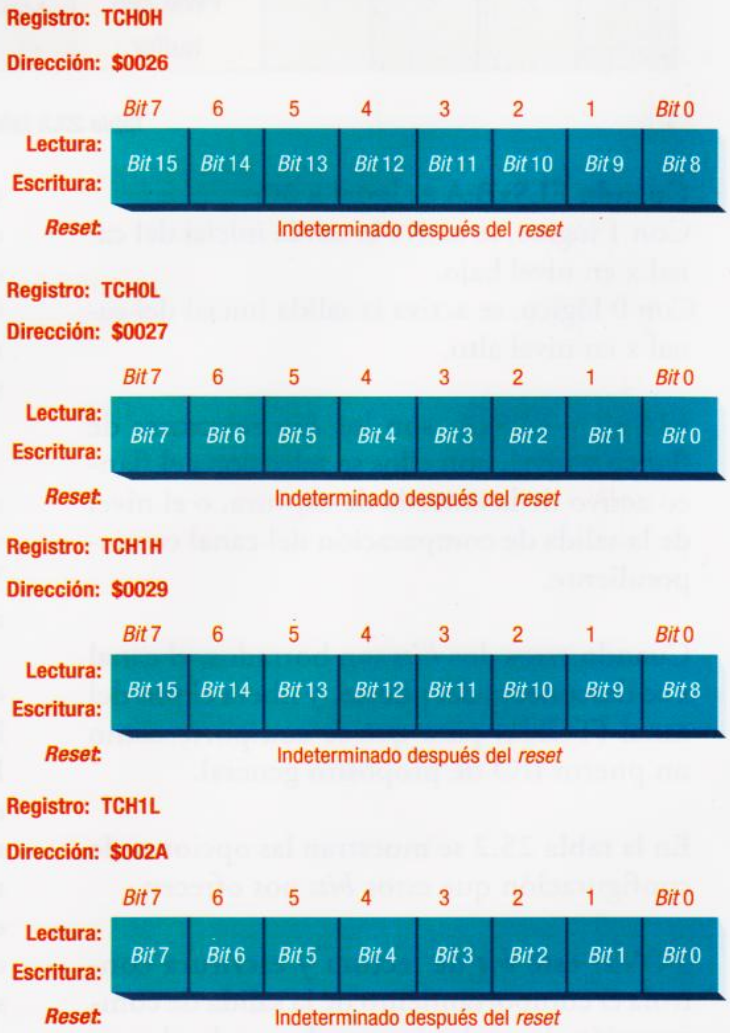
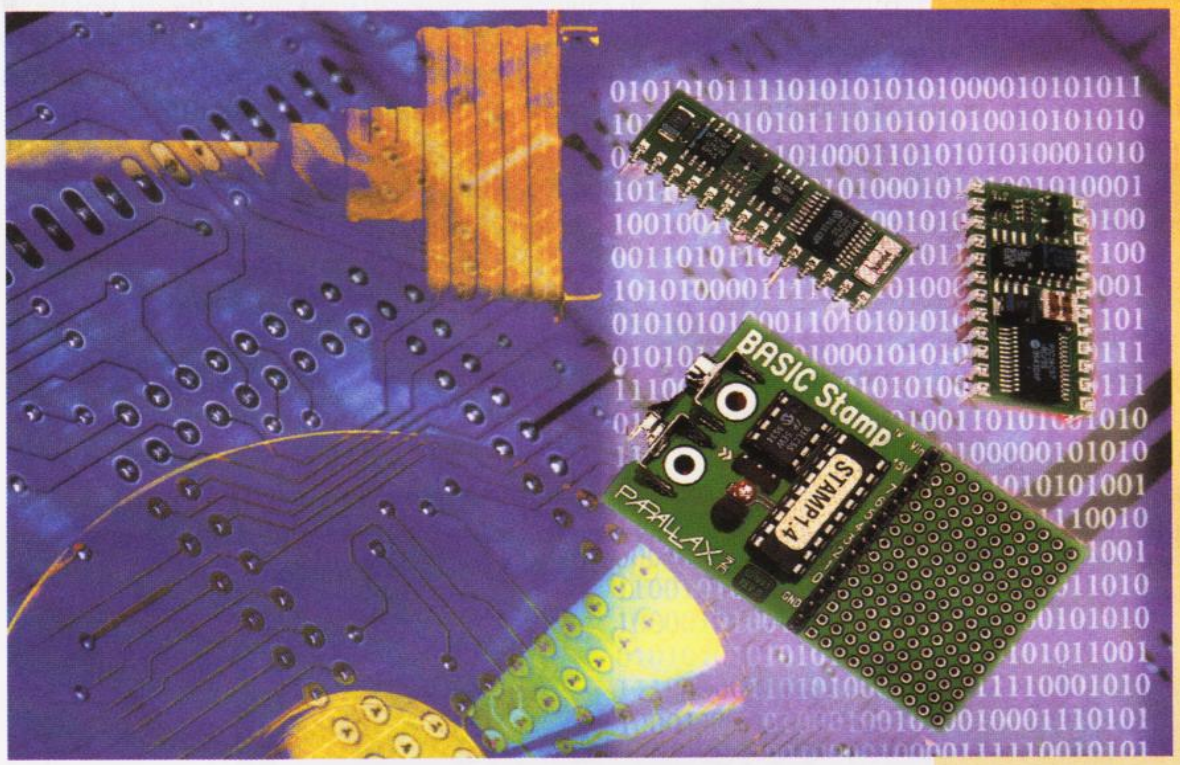


Figura 25.8 *Registros de los canales del TIM (TCH0H/L:TCH1H/L)*

LECCIÓN 26



LOS MICROCONTROLADORES BASIC Stamp

La industria Parallax Inc. ha tomado como base a los microcontroladores PIC y ha desarrollado para ellos una versión simple del lenguaje BASIC, llamada PBASIC, para construir una pequeña y versátil computadora denominada BASIC Stamp.

Gracias a estos dispositivos, los microcontroladores han venido siendo utilizados inclusive por personas que conocen poco o nada de su arquitectura interna, y que no poseen experiencia en el desarrollo de programas en lenguaje ensamblador.

El BASIC Stamp I

En esta lección estudiaremos el microcontrolador BASIC Stamp I (BS1-IC) desarrollado por Parallax Inc. Esta versión de microcontroladores usa la tecnología de montaje superficial y la popularidad del lenguaje de programación BASIC para construir un dispositivo de bajo costo, reducido tamaño, y alta eficiencia, que se puede utilizar como una pequeña computadora de fácil manejo y mucha versatilidad en la gran mayoría de tareas para la industria, el comercio, la educación y el entretenimiento.

El BASIC Stamp I se compone, en cualquiera de sus versiones, de un lenguaje de programación y una estructura de circuitos que lo soporta. El lenguaje de programación se denomina PBASIC,y ha sido desarrollado por Parallax según algunas instrucciones típicas de la versión simple del lenguaje BASIC, tales como: GOTO, FOR...NEXT, IF...THEN, GOSUB, y OUTPUT, complementado con algunas otras instrucciones especiales creadas para facilitar la comprensión y el manejo de las tareas de control y de monitoreo desde el microcontrolador, como por ejemplo: SERIN, PWM, BUTTON, COUNT, DTMFOUT, y PULSIN.

El *hardware* se instala sobre un pequeño circuito impreso de montaje superficial y en él se incluyen: un microcontrolador PIC de Microchip con el programa residente (*firmware*) del intérprete de las instrucciones, una memoria de acceso serial tipo EEPROM, un regulador de +5V, un oscilador, y un circuito de *reset* automático a la desconexión (*Brownout*).

Para programar el BASIC Stamp I el usuario sólo debe tener conocimientos del lenguaje de programación BASIC y disponer de algunos requerimientos mínimos en su equipo; éstos son: el *software* adecuado, una computadora con sistema operativo MS-DOS 2.0 ó superior, 128Kbytes disponibles en memoria RAM, y el puerto paralelo libre.

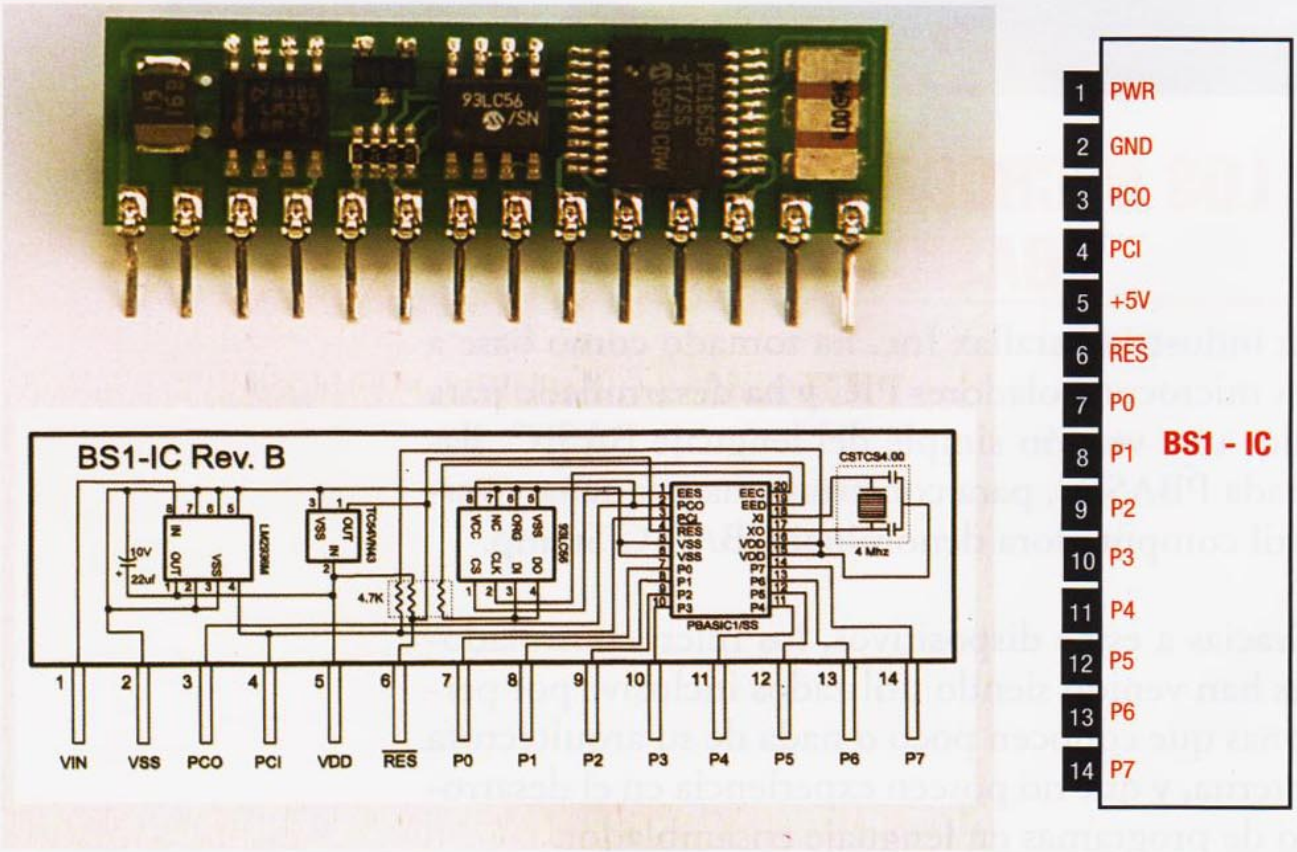


Figura 26.1 Aspecto físico, distribución de los componentes y de los pines del BASIC Stamp I

Distribución de pines del BASIC Stamp I

La versión del BASIC Stamp BS1-IC, que se muestra en la **figura 26.1**, se compone de catorce pines y se fabrica en estructura SIP (empaquetado de una sola fila) para disminuir aún más el espacio ocupado por el dispositivo y facilitar su acople en circuitos impresos y tableros de conexiones (*protoboards*). La asignación y las funciones definidas para los pines de este microcontrolador se describen a continuación:

Pin 1 VIN (PWR): por medio de este pin el microcontrolador recibe la fuente de alimentación de voltaje no regulada, que puede estar entre 6 y 15VCC para esta versión, y la regula a +5VCC por medio de un regulador interno, para alimentar al microcontrolador y al resto del circuito. Cuando esta entrada se alimenta se pueden extraer +5VCC por el pin 5 del BASIC Stamp con el objeto de alimentar algunos circuitos externos.

Pin 2, GND: este pin es la tierra del sistema y debe estar conectado con la tierra de la PC (a través del pin 25 del puerto paralelo) para programar correctamente el microcontrolador.

Pin 3, PC Out (PCO): este pin conecta la señal de control de transferencia de datos, *BUSY* (ocupado), del puerto paralelo de la PC con el microcontrolador.

Pin 4, PC In (PCI): este pin conecta el terminal para la transferencia de datos *DO* del puerto paralelo de la PC con el microcontrolador para programar el dispositivo.

Pin 5, +5V: esta entrada se ha dispuesto en el BASIC Stamp I para conectar una fuente de voltaje regulada entre 4.5VCC y 5,5VCC para alimentar el sistema. Si el circuito se alimenta por este pin, no se debe utilizar la entrada de alimentación V_{IN} (PWR).

Pin 6, $\overline{RES}$: el circuito del microcontrolador usa este pin como la entrada de *reset* activa en nivel bajo de todo el sistema, ésta puede ser colocada en

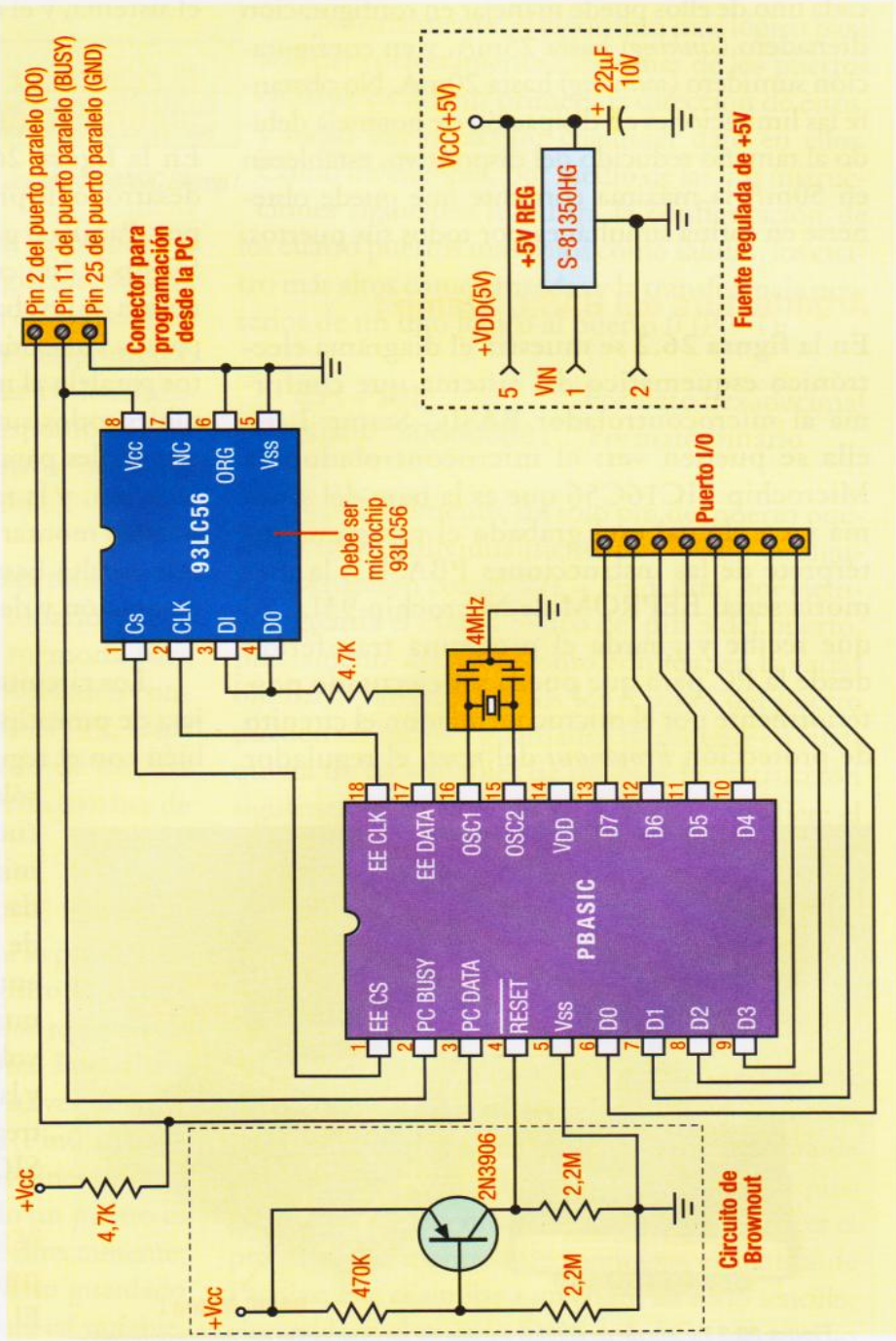


Figura 26.2 *Arquitectura del microcontrolador BASIC Stamp I*

cero lógico para activarlo; pero no se debe poner en nivel alto, porque afecta el comportamiento del circuito de *brownout* que se encarga de generar un *reset* cuando la alimentación cae por debajo de 4V, con el objeto de proteger el *software* del dispositivo.

Pines 7a 14, P0 a P7: estos ocho pines son los puertos de entrada o salida del microcontrolador; cada uno de ellos puede manejar en configuración drenadero (*sinking*) hasta 25mA, y en configuración sumidero (*sourcing*) hasta 20mA. No obstante las limitaciones en disipación de potencia debido al tamaño reducido del dispositivo, establecen en 50mA la máxima corriente que puede obtenerse en forma simultánea por todos sus puertos.

Arquitectura del BASIC Stamp I

En la **figura 26.2** se muestra el diagrama electrónico esquemático del sistema que conforma al microcontrolador BASIC Stamp I, en ella se pueden ver: el microcontrolador de Microchip PIC16C56 que es la base del sistema y en el cual está grabado el programa intérprete de las instrucciones PBASIC, la memoria serial EEPROM de Microchip 93LC56 que recibe y guarda el programa transferido desde la PC para que pueda ser ejecutado posteriormente por el microcontrolador, el circuito de protección *brownout* del *reset*, el regulador de voltaje integrado de +5V que alimenta todo el sistema, y el circuito de oscilación de 4MHz.

FUENTE DE PODER (volts)	CORRIENTE TOTAL (mA)
5 - 9	50
12	40
25	10
40	2 - 3

Tabla 26.1 *Capacidad del regulador interno en función de la fuente de alimentación*

El regulador interno y la tarjeta de prototipos del BASIC Stamp I

En la **figura 26.3** se muestra la tarjeta para el desarrollo de prototipos (*carrier board*) diseñada por Parallax para el microcontrlador BASIC Stamp I; ésta cuenta con terminales para la conexión de una batería de 9VCC, el conector (*socket*) para la conexión del BASIC Stamp I, un conector paralelo al microcontrolador que deja disponibles todos sus pines, un pulsador de *reset*, los terminales para la conexión del cable de comunicación, y la tarjeta multiusos, sobre la cual se pueden montar pequeños circuitos prototipo, lo que resulta bastante útil en la etapa de experimentación y desarrollo de algún proyecto.

Los circuitos que se construyan sobre la tarjeta de prototipos, pueden ser alimentados también con el regulador interno de +5V, pero esto sólo se puede hacer si se tiene en cuenta que éste también alimenta al microcontrolador, y, que su capacidad está en función de la magnitud de la fuente de alimentación en la entrada (VIN). En la **tabla 26.1** se muestra la relación que hay entre el voltaje de la fuente de alimentación y la corriente máxima que puede entregar el regulador interno del BASIC Stamp I.

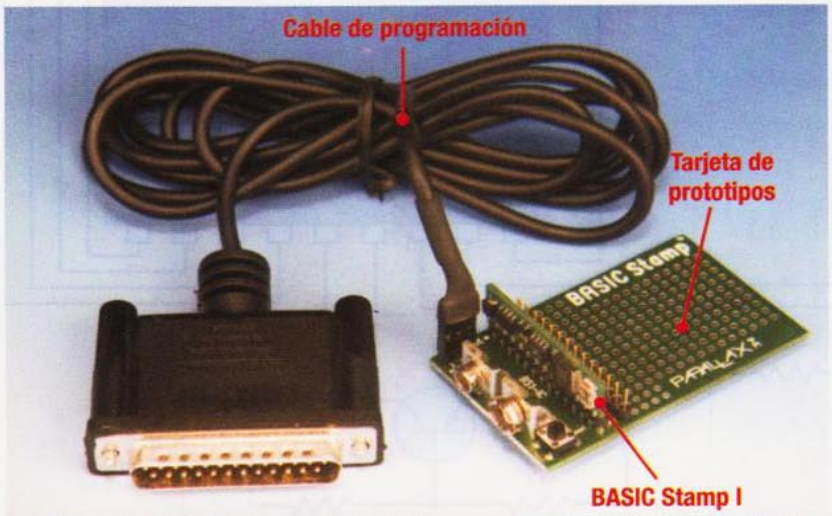


Figura 26.3 *Tarjeta de prototipos y cable de programación del microcontrolador BASIC Stamp I*

Mapa de memoria disponible para el usuario

El BASIC Stamp tiene 256 posiciones de memoria de programa

PALABRA	BYTE	BIT	NOTAS
Puerto	Pins Dirs	Pin0-Pin7 Dir0-Dir7	Pines I/O; direccionables bit a bit Pines de control de dirección I/O; direccionables bit a bit
W0	B0 B1	Bit0-Bit7 Bit8-Bit15	Direccionables bit a bit Direccionables bit a bit
W1	B2 B3		
W2	B4 B5		
W3	B6 B7		
W4	B8 B9		
W5	B10 B11		
W6	B12 B13		Usado por la instrucción GOSUB Usado por la instrucción GOSUB

Figura 26.4 *Mapa de la memoria RAM para el usuario del BASIC Stamp I*

EEPROM y 16 *bytes* de memoria RAM para el usuario, y, por medio de los registros de esta última, se tiene acceso a los pines de los puertos de entrada o salida y a las variables disponibles. En la **figura 26.4** se describen las 16 localidades del mapa de memoria RAM disponibles para el programador, y los pines, registros, nombres y funciones definidas para cada una.

En este microcontrolador, y dependiendo de las necesidades del programa, el usuario puede definir el formato del espacio en memoria para las variables y los puertos I/O, éstos pueden configurarse y accederse *bit* a *bit* (**Pin0** a **Pin7**, **Dir0** a **Dir7**, y ***Bit*0** a ***Bit*15**), en formato de un *byte* (**Pins**, **Dirs**, y **B0** a **B13**), o como una palabra de dos *bytes* (**Port**, y **W0** a **W6**).

Control de los puertos del BASIC Stamp I

Los puertos de este microcontrolador se pueden acceder y controlar por medio del registro **Port**, que es de 16*bits* y está compuesto por dos registros de 8*bits* cada uno, llamados **Pins** y **Dirs**. Por medio del registro Pins que se compone a su vez de los 8 *bits* de entrada o salida de los puertos (**Pin0** a **Pin7**), se pueden leer o escribir datos en los pines de cada puerto independientemente; cuando un puerto es leído, el pin correspondiente se lee directamente; mientras que cuando es escrito, el dato guardado en la RAM, en el espacio definido para tal variable, es el que se transfiere al pin del puerto.

Con el registro Dirs, y a través de sus *bits* (**Dir0** a **Dir7**), que son los *bits* que controlan la dirección de los puertos del microcontrolador, se define la condición del pin del puerto como una entrada o como una salida. Con un "0" puesto en cualquiera de estos *bits*, se define el mismo como una entrada; mientras que con un "1" se configura cualquiera de los pines del puerto como una salida.

Normalmente, el algoritmo más lógico para acceder correctamente a los pines de los puertos consiste en definir primero la dirección de éstos, y luego leer o escribir cualquier dato en ellos. Como un ejemplo, por medio de las dos instrucciones siguientes se ilustra la configuración de los cuatro puertos más bajos como salidas, los cuatro más altos como entradas y la transferencia posterior de un uno lógico al puerto 0 (P0=1):

```
dirs = $0F          'Formato hexadecimal
pins = %00000001    'Formato binario
```

Adicionalmente, un sólo pin del puerto puede ser leído individualmente por medio de alguna instrucción como *Bit*1 = Pin3; ésta, por ejemplo, guarda el estado lógico del pin 3 del puerto, previamente definido como entrada, en la variable *Bit*1. También, todos los 8 pines del puerto pueden ser leídos y guardados fácilmente en cualquiera de las variables de un *byte*; la instrucción siguiente es un ejemplo en el que se transfiere el estado del puerto del microcontrolador (todos sus 8*bits*) a la posición en RAM de la variable B7:

```
b7 = pins
```

Programación del BASIC Stamp I

En lo que a la programación y al buen uso de este microcontrolador se refiere, además de conocer bien las 32 instrucciones del PBASIC y la estructura de programación básica de cualquier lenguaje de programación de alto nivel, el usuario debe conocer el programa de soporte que suministra el fabricante Parallax; éste es similar a un editor de texto sencillo, ejecutable en el entorno DOS de la PC, y, por medio de los comandos de éste, se puede editar, compilar,

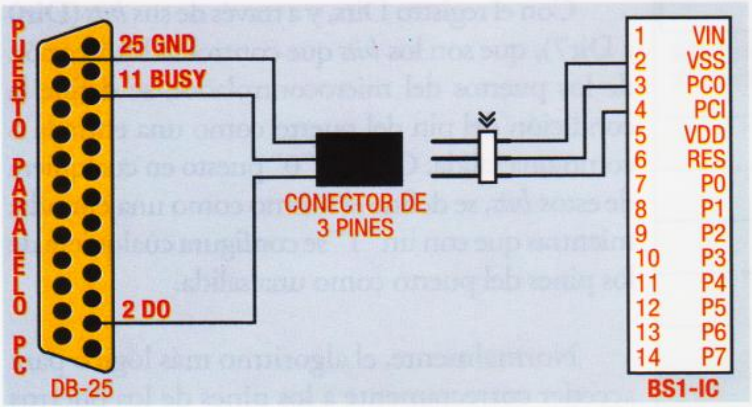


Figura 26.5 *Cable de comunicación para el BASIC Stamp I*

depurar y transferir el programa hacia la memoria EEPROM del sistema BASIC Stamp I, para su posterior ejecución.

Para programar el microcontrolador se usa el puerto paralelo de la computadora, pero para ello se debe usar el cable DB-25 a 3 pines que se suministra especialmente para este propósito por el fabricante; en la **figura 26.5** se muestra la configuración que tiene el cable de comunicación que debe usarse entre la PC y el BASIC Stamp I. Este cable sólo usa tres pines del puerto paralelo de la PC para comunicarse con el microcontrolador, tal como puede verse en el diagrama esquemático de la figura 26.2, y ellos son: el terminal de datos (DO), la señal de control del bus (BUSY), y la tierra (GND).

El BASIC Stamp I opera con niveles TTL, y por lo tanto la salida y entrada de comunicación serial a través de sus puertos, y que se controla por medio de las instrucciones **SERIN** y **SERUOT**, también operan a +5V; este nivel es suficientemente seguro para comunicar eficazmente el microcontrolador con otros dispositivos a distancias cortas (menos de 3m), pero, cuando la comunicación se hace a más de 3m, los datos transmitidos pueden sufrir alteraciones, y por esto se recomienda elevar los niveles de comunicación por medio de una interfaz serial RS-232 estándar tal como el MAX232, el LT1181ACN, o alguna otra similar.

Esto también puede ser necesario en algunas tareas que requieren comunicar el BASIC Stamp I, a través de sus puertos, con el puerto serial de otro dispositivo o equipo, ya que muchas PC cuando operan como equipo receptor no aceptan niveles TTL a través del puerto serial; en la **figura 26.6** se muestra un circuito típico con el LT1181ACN que sirve para cambiar los niveles TTL por niveles de comunicación RS-232.

Conjunto de instrucciones resumidas del lenguaje PBASIC

El lenguaje de programación del BASIC Stamp I está compuesto por un conjunto reducido de instrucciones (32 instrucciones) de alto nivel, y se denomina PBASIC; éste ha sido diseñado por Parallax según una versión simple del popular programa BASIC. Con este programa el diseñador cuenta con muchas herramientas que son di-

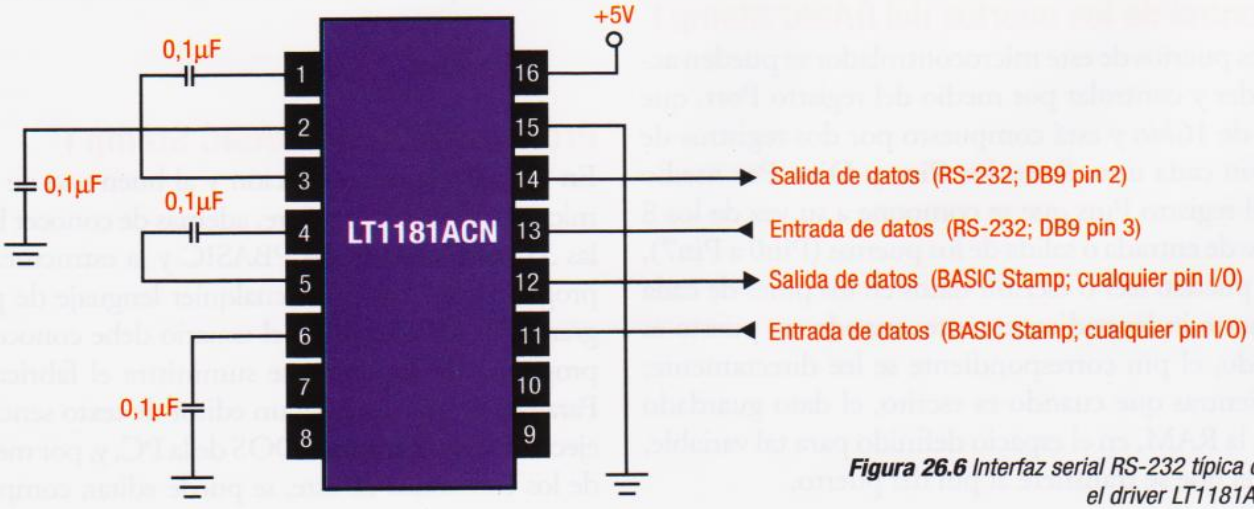


Figura 26.6 *Interfaz serial RS-232 típica con el driver LT1181ACN*

fíciles de construir usando el lenguaje ensamblador tradicional de los microcontroladores y que son de frecuente uso en las aplicaciones de estos dispositivos en cualquier campo; algunas de estas rutinas diseñadas como herramientas y que se activan por medio de las instrucciones especiales de este lenguaje, son: generación de sonidos, lectura directa y sin rebote de pulsadores e interruptores, lectura de resistencias variables, comunicación serial RS-232, generación y medición de tiempos, y salida PWM, entre otras.

En la **tabla 26.2** se muestra el conjunto de 32 instrucciones del lenguaje PBASIC que soporta al microcontrolador BS1-IC y se hace una descripción breve de la o las funciones de cada una.

CONJUNTO DE INSTRUCCIONES RESUMIDAS DEL BASIC Stamp I	
SALTOS	
IF....THEN	Bifurcación condicional
BRANCH	Salta a la dirección especificada por el offset
GOTO	Salta a una dirección
GOSUB	Salta a la subrutina direccionada. Se permiten hasta 16 GOSUB
RETURN	Retorno desde subrutina
CICLOS	
FOR....NEXT	Establece un ciclo de este tipo Numéricas
{LET}	Permite manipulación de variables como A=5, B=A+2, etc. Posibilita operaciones como suma, resta, multiplicación, división, límite máximo, límite mínimo y operaciones lógicas como AND, OR, XOR, NAND, NOR y XNOT.
LOOKUP	Instrucción para manejo de tablas. Permite el manejo de datos mediante offset y su almacenamiento en variables específicas
LOOKDOWN	Busca números iguales (de 0 a N) y lo almacena en variables
RANDOM	Genera un número seudoaleatorio
ENTRADAS Y SALIDAS DIGITALES	
OUTPUT	Configura un pin como salida
LOW	Pone un pin de salida en nivel bajo
HIGH	Pone un pin de salida en nivel alto
TOGGLE	Configura un pin como salida e invierte su estado lógico
PULSOUT	Genera un pulso de salida invirtiendo el estado lógico de un pin por algún tiempo
INPUT	Configura un pin como entrada
PULSIN	Mide un pulso de entrada
REVERSE	Si un pin está configurado como salida, lo pone como entrada y viceversa.
BUTTON	Hace el antirebote de un botón y puede hacer un salto a una dirección si así se desea.
ENTRADAS Y SALIDAS DIGITALES SERIALES	
SERIN	Entrada serial de datos. Posee calificadores y variables que, si se cumplen, permiten el almacenamiento de los datos recibidos. Las velocidades permitidas son 300, 600, 1.200 y 2.400 bps. Los datos deben cumplir con la especificación de no paridad, datos de 8 bits y 1 stop bit.
SEROUT	Envía datos serialmente a 300, 600, 1.200 ó 2.400 bps, con el mismo formato que se acaba de describir.
ENTRADAS Y SALIDAS ANÁLOGAS	
PWM	Entrega una señal PWM y después configura el pin como una entrada. Se usa para generar voltajes (de 0 a 5) con un condensador y una resistencia.
POT	Lee un potenciómetro de 5 a 50 Kohm y escala el resultado.
SONIDO	
SOUND	Genera algunas notas. La nota 0 es silencio, notas 1 a 127 son tonos ascendentes, y notas 128 a 255 son ruidos blancos.
ACCESO A LA EEPROM	
EEPROM	Almacena datos en EEPROM antes de descargar el programa BASIC.
READ	Lee un byte de EEPROM y lo almacena en una variable
WRITE	Escribe un byte en EEPROM
TIEMPO	
PAUSE	Detiene la ejecución por un período de 0 a 65.536 milisegundos
CONTROL DE POTENCIA	
NAP	Entra en modo de bajo consumo por un corto período
SLEEP	Entra en modo sleep o de bajo consumo por un tiempo de 0 a 65.536 segundos.El consumo de potencia se reduce aproximadamente a 20 µA.
END	Entra en modo sleep hasta que cambia el ciclo de control de potencia o se conecte la computadora
DEPURACIÓN	
DEBUG	Envia variables a la computadora para que sean vistas

Tabla 26.2 *Resumen del conjunto de instrucciones del lenguaje PBASIC para el microcontrolador BASIC Stamp I*

PARA SABER MÁS:
Ejemplo de programación con PBASIC

Por medio de este programa diseñado a propósito para exhibir la fortaleza y el uso apropiado de dos de las instrucciones especiales de este lenguaje, también se ilustra la eficiencia y facilidad de uso del mismo. En este caso se pretende construir un circuito que detecte la intensidad de la luz y lo informe por medio de un tono audible variable; esto se hará por medio de un pequeño parlante piezoeléctrico (con alta impedancia, más de 40 Ω) conectado al pin del puerto uno, y la intensidad de la luz recibida se detectará a través de una fotocelda (con resistencia entre 5k y 50k) conectada en el puerto cero.

Para esto se leerá en todo momento el valor de la resistencia en los terminales de la fotocelda por medio del puerto P0, y, de acuerdo con su magnitud, se cambiará la frecuencia del tono y se entregará al parlante por el puerto P1. En este caso se usarán las instrucciones especiales **POT** y **SOUND**, que están diseñas para leer resistencias variables y generar tonos, respectivamente.

El programa que nos permite este desarrollo se muestra y comenta a continuación:

```
repetir:                 'Define la etiqueta "repetir".

    pot 0,100,b2         'Lee la resistencia de la fo-
                         'tocelda en el pin del puerto
                         '0, y guarda la lectura en la
                         'variable b2.

    b2=b2/2              'Divide el resultado de la
                         'lectura por 2, de forma tal
                         'que no exceda nunca de 128.

    sound 1,(b2,5)       'Genera un tono por el pin
                         'del puerto 1, y en él, la
                         'frecuencia depende del
                         'valor en b2 y el tiempo de
                         'duración del tono es 5 x
                         '12ms = 60ms.

    goto repetir         'Salta incondicionalmente a
                         'repetir el ciclo.
```

Observando este programa resultan claros los beneficios que se obtienen al trabajar con un lenguaje de alto nivel y un programa interprete (*firmware*), dedicados a la programación de microcontroladores; el reducido número de instrucciones usadas y un código de programa amigable, poderoso y compacto, son su principal ventaja.

En la **figura 26.7** se muestra el diagrama esquemático simple que se requiere para montar este proyecto en la tarjeta de prototipos o en un tablero para el montaje de circuitos electrónicos (*protoboard*).

Para dar inicio la ejecución de las tareas de este programa, solamente se require que éste sea grabado por medio del cable de programación en la memoria EEPROM, y posteriormente, conectar la fuente de alimentación al microcontrolador.

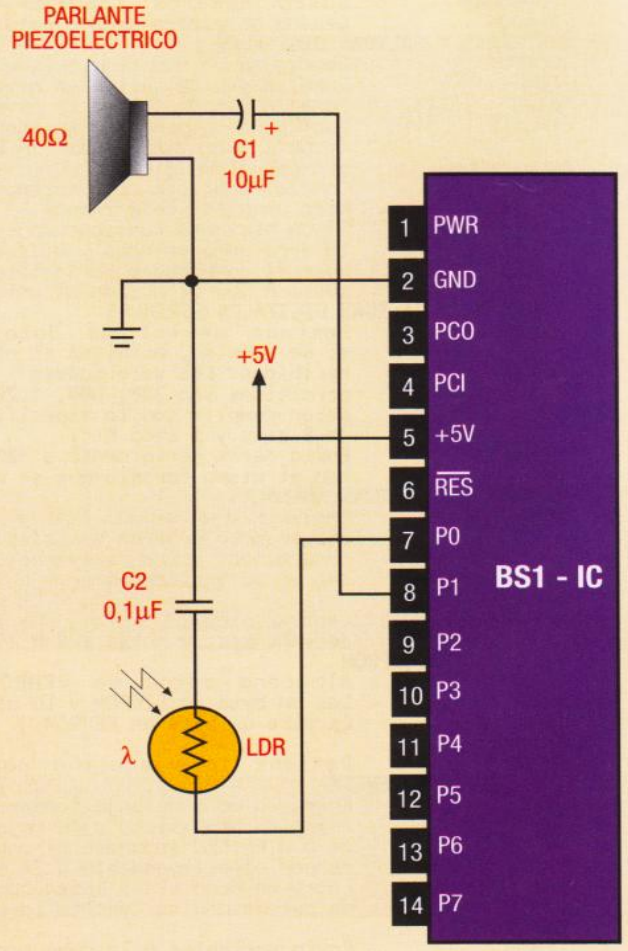
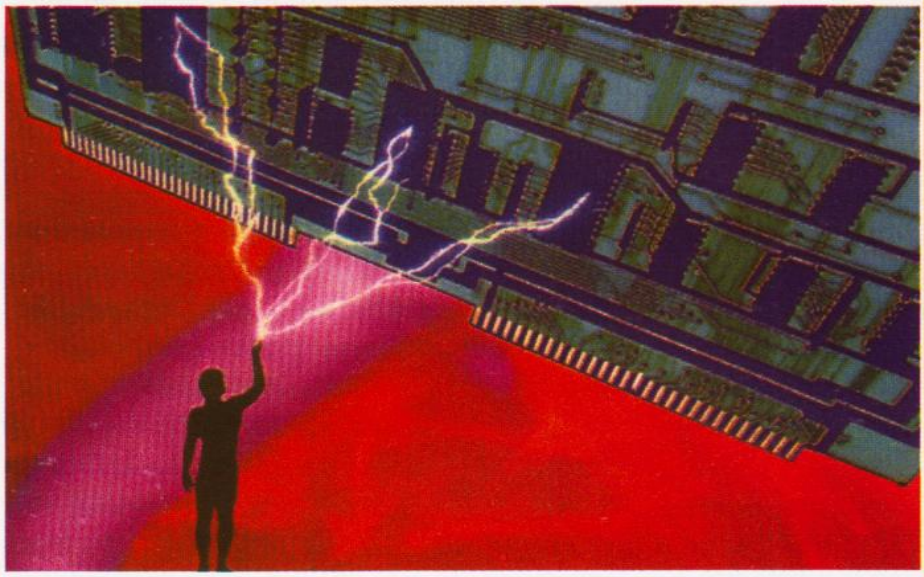


Figura 26.7 *Diagrama esquemático del medidor audible de luz*

LECCIÓN 27



CONJUNTO DE INSTRUCCIONES DE LOS MICROCONTROLADORES BASIC Stamp I

En esta lección se estudiará el conjunto de instrucciones del lenguaje PBASIC de Parallax, Inc. que está dedicado a la programación desde alto nivel de los microcontroladores BASIC Stamp I, este conjunto de instrucciones lo integran 32 de ellas, muy compactas, poderosas y con funciones especializadas en el manejo de las tareas más frecuentes encomendadas a los microcontroladores.

Conjunto de instrucciones del lenguaje PBASIC del BASIC Stamp I

El reducido conjunto de 32 instrucciones del lenguaje PBASIC está compuesto por algunas instrucciones típicas del lenguaje BASIC y de otras especializadas que poseen múltiples funciones. En esta lección las estudiaremos con todos sus interruptores, y para facilitar la ubicación rápida de cualquiera de ellas, se hará en orden alfabético; las instrucciones son:

Nota:
De acuerdo con la nomenclatura para la representación de las instrucciones, se escribirán entre paréntesis los términos obligatorios, entre corchetes los términos opcionales, y los comentarios empezarán con una comilla simple.

1. BRANCH offset,(address0,address1, ...,addressN)

Con esta instrucción se salta a la dirección especificada por el desplazamiento (*offset*), si está dentro del rango.

offset: el desplazamiento es una variable o una constante que especifica la dirección del salto entre 0 y N lugares.
addressX: las direcciones son nombradas por etiquetas que definen hacia donde se debe hacer el salto, se separan por coma y su número define el alcance del desplazamiento.

Esta instrucción es una variación de IF...THEN, así por ejemplo, si se escribe la siguiente línea de comando:

```
BRANCH b3,(caso_0,caso_1,caso_2)
```

se podrá obtener el mismo resultado que al escribir las tres siguientes líneas de comando:

```
IF b3=0 THEN caso_0     'si b3=0 salta a la
                        etiqueta "caso_0"
IF b3=1 THEN caso_1     'si b3=1 salta a la
                        etiqueta "caso_1"
IF b3=2 THEN caso_2     'si b3=2 salta a la
                        etiqueta "caso_2"
```

En cualquier caso, cuando el *offset* este fuera de rango (en este ejemplo, si b3 es mayor que 2), la instrucción BRANCH no será ejecutada.

2. BUTTON pin,downstate,delay,rate,bytevariable, targetstate,address

Con esta instrucción se habilita la rutina de la entrada antirrebotes para interruptores y pulsadores, con autorrepetición, y salto a una dirección cuando el estado leído es el esperado. El circuito externo para el pulsador o interruptor puede ser activo en nivel bajo o en nivel alto, tal como se muestra en la **figura 27.1**.

pin: es una variable o una constante que especifica el pin I/O que se va a usar como entrada de interruptor o pulsador, entre 0 y 7.
downstate: es una variable o una constante que especifica el estado lógico activo, y que será leído al accionar el interruptor o pulsador, éste puede ser 1 ó 0.
delay: es una variable o una constante entre 0 y 255 que especifica cuanto tiempo (en ciclos de esta rutina) debe mantenerse activado el interruptor o el pulsador, antes de iniciar la autorrepetición o lectura sucesiva del pin definido. Los dos valores extremos 0 y 255 son especiales; cuando se pone 0 la rutina devuelve el estado de la entrada, pero no ejecuta el antirrebote ni la autorrepetición; mientras que con 255, si se ejecuta el antirrebote.

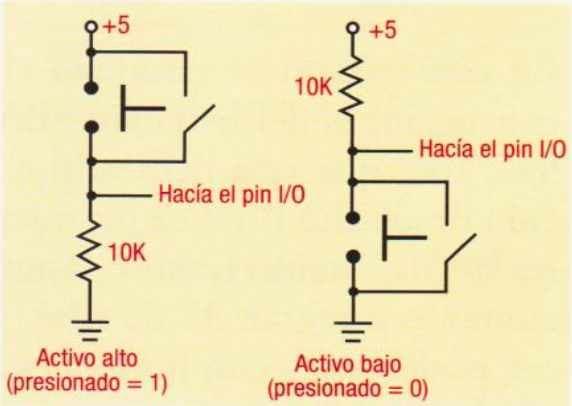


Figura 27.1 *Conexión externa de los interruptores o pulsadores en sus dos estados posibles activo-alto y activo-bajo*

rate: es una variable o una constante que define la velocidad de las lecturas sucesivas y se expresa entre 0 y 255 ciclos de esta rutina.
bytevariable: es el área de trabajo (cualquier registro de trabajo de un *byte*) de esta rutina. La variable elegida debe ser borrada con 0 lógico antes de usarla por primera vez desde la rutina de BUTTON.
targetstate: es una variable o una constante que especifica el estado lógico en que debe estar el pulsador o interruptor (0=libre, 1=presionado) para que el salto ocurra.
address: es una etiqueta con la que se indica a donde se debe saltar si el targetstate se cumple.

3. DEBUG variable{,variable}

Con esta instrucción se puede monitorear el curso del programa en cualquier momento, por medio de ella se puede visualizar en la pantalla de cualquier PC conectada con el BASIC Stamp I el estado de un bit, un byte, o una palabra. Con las siguientes líneas de comando, a manera de ejemplo, se ilustran las opciones de esta instrucción:

DEBUG b4	'muestra "b4=", 'seguido por el 'valor de b4
DEBUG #b4	'muestra el valor 'de b4
DEBUG "la lectura es",b4	'muestra "la lectu- 'ra es", seguido 'por el valor de b4
DEBUG #%b4	'muestra el valor 'de b4 en binario
DEBUG #@b4	'muestra el carác- 'ter ASCII que es 'equivalente al 'valor de b4
DEBUG"entradas=",b3,b4,cr	'muestra los carac- 'racteres "entra- 'das =", seguidos 'por los valores de 'b3 y b4, y retor- 'na el cursor

Esta instrucción también acepta algunos comandos propios del manejo de la pantalla de la PC, tal como se muestra en la última línea de comando del ejemplo anterior, éstos son: CR (*carriage return*) para el retorno del cursor al comienzo de la línea, y CLS (*clear screen*) para borrar la ventana de visualización. Aquí también se puede ver que lo que se escriba entre comillas será mostrado sin cambio y que por medio de comas se deben separar las variables diferentes en una línea de comandos.

4. EEPROM {location},(data,data,...)

Con esta instrucción se pueden guardar varios datos en la memoria EEPROM del sistema que luego serán utilizados por el programa, o que no deban ser modificados.

location: la ubicación es una variable o una constante opcional entre 0 y 255 que nos indica la posición inicial de la EEPROM a partir de la cual serán guardados los datos que sean enviados. Si no se escribe una ubicación inicial, la posición inicial para la escritura de los datos será la próxima disponible en la memoria.
data: los datos son las variables o constantes expresadas en 8*bits*, y que serán guardadas en localidades consecutivas de la memoria a partir de la posición de inicio.

5. END

Con esta instrucción el microcontrolador entra en adormecimiento (modo *sleep*) y sólo despierta al conectar la PC, o durante los ciclos de potencia. En este estado, y asumiendo que el BASIC Stamp I no está manejando carga por sus puertos, la corriente consumida se reduce a cerca de 20µA.

Es importante considerar el caso en el cual se ejecuta la instrucción END mientras algún puerto configurado como salida está activo en cualquier nivel (1 ó 0) y está manejando una carga; en esta situación el sistema continuará suministrando la corriente de la carga desde la

fuente interna regulada, pero cada 2,3s interrumpirá la corriente a través de todos los puertos por cerca de 18ms. Esto se debe a que el temporizador del *watchdog* propio del *chip* intérprete y que es inalterable, genera un *reset* cada 2,3s que despierta al microprocesador de su adormecimiento y configura a todos sus pines como entrada; esta condición se mantiene por cerca de 18ms que es el tiempo que tarda el *firmware* del PBASIC en tomar el control, restablecer el estado de los pines, y poner a dormir nuevamente el BASIC Stamp I.

Por lo anterior, y si se va a usar esta instrucción, asegúrese de ejecutarla en el momento apropiado, o de conectar cargas que puedan soportar estas desconexiones, o que tengan una interfaz que las admita.

6. FOR variable = start TO end {STEP {-} increment}...NEXT {variable}

Con esta instrucción se implanta un lazo del tipo FOR...NEXT limitado a ocho anidamientos como máximo.

variable: es una variable que se usa como contador interno para limitar el comienzo y el final del ciclo. Con variables tipo *bit* se puede contar de 0 a 1, con tipo *byte* desde 0 a 255, y con tipo *word* desde 0 hasta 65.535.
start: es una variable o una constante que especifica el valor inicial de la variable del contador.
end: es una variable o una constante que especifica el valor final de la variable del contador.
increment: es una variable o una constante opcional que determina el incremento o decremento del contador del ciclo; si no se define un valor particular para el paso (*step*) la variable incrementará en uno, mientras que si se pone el signo (-) decrementará en uno.
variable (después de NEXT): es opcional y se usa para aclarar a cual ciclo FOR...NEXT se refiere el NEXT en cuestión, cuando hay varios de estos ciclos anidados.

7. GOSUB address

Esta instrucción es para un salto incondicional a una subrutina, al ejecutarla, el curso del programa se desvía hacia a la dirección especificada y continua desarrollandose a partir de allí hasta que encuentra la instrucción de RETURN. Antes de saltar el sistema guarda la dirección de la próxima instrucción para continuar la ejecución del programa desde el mismo punto, al retornar de la subrutina.

address: es la etiqueta con la que se nombra la dirección para el salto. Sólo se aceptan hasta 16 llamados a subrutinas, y hasta 4 anidamientos.

8. GOTO address

Esta instrucción es un salto incondicional a cualquier parte del programa, al ejecutarla, el curso del programa se desvía hacia a la dirección especificada y continua su ejecución a partir de allí.

address: es la etiqueta con la que se nombra la dirección para el salto.

9. HIGH pin

Con esta instrucción se pone en nivel alto el pin especificado, y además, si éste ha sido configurado previamente como una entrada, será reconfigurado como una salida.

pin: es la variable o la constante con la que se especifica el pin de salida alta, y de acuerdo con el número de puertos disponibles puede estar entre 0 y 7.

10. IF variable ?? value {AND/OR variable ?? value...} THEN address

Con esta instrucción de salto condicionado se comparan una o varias variables con un valor o valores y se salta a una dirección definida si el resultado es verdadero.

??: es uno de los siguientes operadores matemáticos: = (igual), <> (diferente), > (mayor

que), < (menor que), >= (mayor ó igual que), y <= (menor ó igual que).
variable: es la variable que se compara con el valor o valores.
value: es una variable o una constante que representa el valor para comparación.
address: es una etiqueta con la que se nombra la dirección a donde se debe saltar si el resultado de la comparación o comparaciones es verdadero.

11. INPUT pin

Con esta instrucción se convierte el pin especificado en una entrada, sin importar cual sea su condición previa o el estado en que se estén poniendo los puertos debido a dispositivos externos.

pin: es la variable o la constante con la que se especifica el pin que se va a usar como entrada, y puede estar entre 0 y 7.

12. {LET} variable = {-}value ?? value...

Con esta instrucción se asigna un valor a una variable y también se pueden efectuar operaciones lógicas entre los valores para definir el valor que se asignará. El mnemónico de la instrucción (LET) puede omitirse; por ejemplo, la instrucción "J=7" es equivalente a "LET J=7".

OPERADORES DE LA INSTRUCCIÓN LET

Símbolo	Nombre
+	Suma
-	Resta
*	Multiplicación (devuelve la palabra baja del resultado)
**	Multiplicación (devuelve la palabra alta del resultado)
/	División (devuelve el cociente)
//	División (devuelve el residuo)
min	Mantiene una variable mayor o igual que un valor
max	Mantiene una variable menor o igual que un valor
&	AND lógica
|	OR lógica
^	XOR lógica
&/	NAND lógica
|/	NOR lógica
^/	XNOR lógica

Tabla 27.1 *Tabla de operadores aceptados por la instrucción de asignación LET*

??: puede ser alguno de los operadores que se muestran en la **tabla 27.1**.
variable: es la variable a la que se le asigna el valor directo, o el que resulta de la operación definida en la propia instrucción.
value(s): es la variable o la constante que se tiene en cuenta para calcular el valor que será asignado a la variable.

En cualquier caso se debe tener en cuenta que las operaciones llevadas a cabo por esta instrucción entre los valores están limitadas por el manejo que el PBASIC le da a las variables y a las operaciones matemáticas.

13. LOOKDOWN target,(value0,value1, ...valueN),variable

Esta instrucción se utiliza para buscar uno o varios valores dentro de una lista, en función de un valor objetivo. Cuando el valor objetivo coincide con alguno de los valores de la lista, se guarda la posición que dicho valor tiene dentro de la lista (0 a N) en la variable.

target: es el valor de la variable o la constante que está siendo explorada (el valor objetivo).
value0 a valueN: es una lista de valores con la que se compara el valor objetivo.
variable: es la variable en la que se guarda el resultado de la búsqueda.

Como ejemplo para aclarar la estructura y la función de esta instrucción se escribe la siguiente línea de comando con comentarios, y se supone que la variable b3 se carga en cualquier momento con la letra que representa a uno de los tres comandos siguientes: giro (**G**), avance (**A**) y paro (**P**).

```
LOOKDOWN b3,("G","A","P"),b4
                'si b3="G" entonces b4=0
                'si b3="A" entonces b4=1
                'si b3="P" entonces b4=2
```

14. LOOKUP offset,(value0,value1,...valueN), variable

Esta instrucción se utiliza para buscar un valor especificado por un *offset* dentro de los datos de una lista y guardarlo en una variable. Esto quiere decir que si los elementos de una tabla son: 3,5,7 y 9; y el *offset* es 2, la variable contendrá el número 7.

offset: es un número entre 0 y 255 que se usa como índice para buscar el valor.
value0 a valueN: es una tabla de hasta 256 datos de 8 *bits*.
variable: es la variable en la que se guarda el resultado de la búsqueda.

15. LOW pin

Con esta instrucción se pone en nivel bajo un pin especificado. Si el pin ha sido previamente definido como una entrada, esta instrucción lo reconfigurará como una salida.

pin: es una variable o una constante entre 0 y 7 con la cual se especifica el pin I/O del puerto que se va a utilizar.

16. NAP period

Con esta instrucción se pone a dormir el microprocesador por un periodo corto de tiempo y se reduce el consumo de potencia hasta en el 90%. En la **tabla 27.2** se resumen los ocho periodos que pueden configurarse.

period: es una variable o una constante con la que se determina la duración del tiempo de bajo consumo o de adormecimiento NAP. El periodo puede definirse entre 0 y 7, y el tiempo de NAP está determinado por la siguiente ecuación:

$$\text{Tiempo de NAP} = 18 \times (2^{\text{periodo}})$$

Cuando se usa esta instrucción todas las cargas conectadas en los pines de los puertos configurados como salidas continúan siendo alimentadas por el BASIC Stamp I, pero es importante recordar que al final de cada periodo de NAP el microcontrolador despierta debido a un *reset*, y que éste

Período	2 Período	Duración del NAP
0	1	18 ms
1	2	36 ms
2	4	72 ms
3	8	144 ms
4	16	288 ms
5	32	576 ms
6	64	1.152 ms
7	128	2.304 ms

Tabla 27.2 *Tabla de selección de los periodos de duración del NAP*

configura todos los pines como entrada durante 18ms; lo que ocasiona una interrupción de la corriente a través de dichas cargas. Con el siguiente programa, observando la salida en el osciloscopio o el parpadeo en un diodo LED, tal como se muestra en la **figura 27.2**, se puede demostrar dicho comportamiento.

```
        LOW 7          'pone en cero la salida del
                       'pin 7
Repetir:
        NAP 2          'pone a dormir al mi-
                       'crocontrolador durante
                       '72ms
        GOTO Repetir   'salta a ejecutar nueva-
                       'mente la instrucción
                       'NAP 2
```

17. OUTPUT pin

Por medio de esta instrucción se configura como una salida cualquier pin del puerto que se especifique.

pin: es la variable o la constante con la que se especifica el pin que se va a configurar como

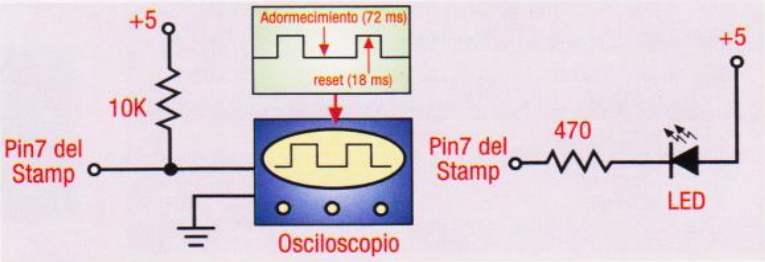


Figura 27.2 *Circuitos que se pueden usar para observar el efecto del despertar del periodo de NAP, de acuerdo con el anterior programa*

salida, y de acuerdo con el número de puertos disponibles puede estar entre 0 y 7.

Es importante recordar que, al comenzar a ejecutar un programa o luego del *reset*, todos los pines del puerto son configurados como entradas por el sistema automáticamente, y, por lo tanto, el usuario debe configurar como salidas aquellos que necesite.

18. PAUSE milliseconds

Usando esta instrucción se detiene la ejecución del programa durante el número de milisegundos especificados.

milliseconds: es la variable o la constante con la que se determina la cantidad de milisegundos durante los cuales se va a detener la ejecución del programa. La pausa puede estar entre 0 y 65.535ms, y, puesto que usa el circuito oscilador del BASIC Stamp I como base de tiempo, su precisión es la misma, ±1%.

Al usar la pausa tenga en cuenta que el programa intérprete del sistema sólo puede ejecutar hasta 2.000 instrucciones por segundo y que esta velocidad relativamente baja puede ocasionar problemas si se detiene la ejecución en momentos críticos, y en ciertas tareas que dependen considerablemente del tiempo.

19. POT pin,scale,variable

Usando esta instrucción se puede leer cualquier tipo de resistencia variable entre 5 y 50kW, como por ejemplo, un potenciómetro, un reóstato, un termistor, o una fotorresistencia (LVR), entre otras. Un extremo de la resistencia se debe conectar con el pin del puerto, mientras que el otro lado se conecta con un condensador a tierra, tal como se aprecia en el circuito típico de la **figura 27.3**.

Esta instrucción configura el pin especificado como una salida y calcula indirectamente el valor de la resistencia variable en un instante,

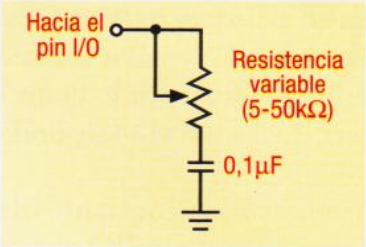


Figura 27.3 *Circuito típico para la adecuada conexión y lectura de una resistencia variable desde el BASIC Stamp I*

para esto el microcontrolador mide el tiempo que tarda la descarga del condensador a través de dicha resistencia.

pin: es la variable o la constante (de 0 a 7) en la que se especifica el pin I/O en donde se ha conectado la resistencia.
scale: es la variable o la constante (de 0 a 255) que se usa para definir el factor de escala del resultado de la conversión, el cual internamente siempre tiene una resolución de 16*bits*. La lectura siempre es multiplicada por la siguiente relación:

Resultado = (lectura 16 bits x scale) /255

variable: es la variable que se usa para almacenar el resultado final de la lectura escalada a un valor de 8*bits*.

Con el siguiente programa, a manera de ejemplo, se lee continuamente el valor de un potenciómetro conectado en P3 y el resultado se envía al exterior por medio de la salida serial P4.

```
Leerpot:
    POT 3,170,b7          'lee el potenció-
                          'metro del pin 3
                          'en b7
    SEROUT 4,N300,(b7)    'envía la lectura
                          'serialmente por
                          'el pin 4
    GOTO Leerpot          'va a leer nueva-
                          'mente
```

20. PULSIN pin,state,variable

Por medio de esta instrucción se convierte el pin especificado en una entrada y se mide un pulso

de entrada en unidades de 10ms. Esta instrucción siempre usa el contador interno del sistema que es de 16*bits*, y por lo tanto la máxima medida será igual a 0,65534 segundos.

pin: es la variable o la constante (de 0 a 7) en la que se especifica el pin I/O que se va a usar.
state: es una variable o una constante que puede ser 0 ó 1 y con la cual se define tipo de flanco a partir del cual se va a iniciar la medición del pulso, éste puede ser descendente o ascendente respectivamente.
variable: es la variable que se usa para depositar el resultado de la medición. Ésta debe ser una palabra de 16 *bits* (W0 a W6) para que pueda albergar el contenido completo del contador interno que puede contar entre 1 y 65.534 unidades de tiempo (10ms), o de 8 *bits* (B0 a B13) para que pueda contener el registro más bajo del contador, con el que puede contar entre 1 y 255 unidades de tiempo.

21. PULSOUT pin,time

Por medio de esta instrucción se puede generar un pulso durante un tiempo especificado. Este pulso se produce invirtiendo el estado lógico previo del pin que se haya definido como salida del pulso.

pin: es la variable o la constante (de 0 a 7) en la que se especifica el número del pin I/O del microcontrolador por el cual se va a generar el pulso.
time: es la variable o la constante en la que se especifica la longitud del pulso en unidades de tiempo de 10ms. Puede estar entre 0 y 65.535 unidades de tiempo, lo que le permite al usuario generar un pulso de hasta 0,65535 segundos.

22. PWM pin,duty,cycles

Usando esta instrucción se puede producir una señal modulada por ancho de pulsos (PWM) sobre cualquiera de los pines I/O del puerto. Cuando la salida PWM termina, el microcontrolador retorna automáticamente dicho pin

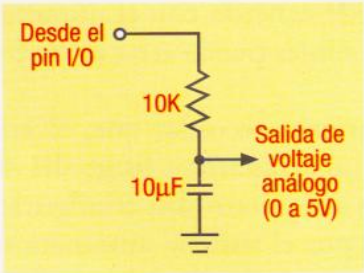


Figura 27.4 *Circuito típico para generar un voltaje análogo por medio de la salida de PWM*

a la condición de entrada, y esto nos permite usar esta modulación para generar un voltaje análogo entre 0 y 5V usando un circuito con una resistencia y un condensador a tierra conectados al pin de salida, tal como se muestra en la **figura 27.4**; en este circuito el voltaje análogo se obtiene en terminales del condensador, y la señal PWM debe ser generada continuamente para actualizar o refrescar el nivel de la carga del condensador.

pin: es la variable o la constante (de 0 a 7) en la que se especifica el número del pin I/O del microcontrolador por el cual se va a generar la modulación PWM.
duty: es una variable o una constante entre 0 y 255 con la cual se especifica el nivel de voltaje esperado, entre 0 y 5V respectivamente.
cycles: es una variable o una constante entre 0 y 255 con la cual se determina la cantidad de ciclos de salida. Cada ciclo de salida se tarda 5ms y el número de ellos que se requieren está en función del tamaño del condensador, los condensadores más grandes necesitan de un mayor número de ciclos para alcanzar su plena carga (5V).

La salida PWM emite una ráfaga de ceros y unos con una relación determinada por el valor de la variable *duty*, si ésta es 255, entonces la salida será un nivel alto continuo, pero si es 0, la salida será un nivel bajo permanente. Para todos los valores del ciclo de servicio (*duty*), que estén entre 0 y 255, la relación de los pulsos estará dada por la expresión ***duty*/255**; por ejemplo, si *duty* es 50 la relación de unos a ceros será aproximadamente del 19,6% (50/255 = 0,196).

Cuando la salida PWM se utiliza para cargar un condensador y obtener entre sus terminales un voltaje análogo, la magnitud de éste se obtiene a partir de la expresión 5*(*duty*/255); por ejemplo, si *duty* es 200 el voltaje que alcanzará el condensador será aproximadamente igual a 3,92V (5*(200/255)).

Para obtener los mejores resultados de esta salida, no afectar la modulación y reducir el número de ciclos de actualización necesarios, la corriente que de tal pin se extraiga debe ser mucho menor que 1µA; para esto se recomienda la conexión de un *buffer* de alta impedancia entre el pin de salida del microcontrolador y la carga, tal como el seguidor de voltaje de precisión construido con el operacional CA5160 que se muestra en la **figura 27.5**.

23. RANDOM wordvariable

Usando esta instrucción se puede generar un número seudo-aleatorio en una variable tipo *word*.

wordvariable: es una variable de 16*bits* que puede estar entre 0 y 65.535, y en la cual se encuentra el número seudo-aleatorio que resulta de la ejecución de esta instrucción.

24. READ location,variable

Utilizando esta instrucción se puede leer cualquier posición de la memoria EEPROM del sistema y guardar el valor en una variable determinada.

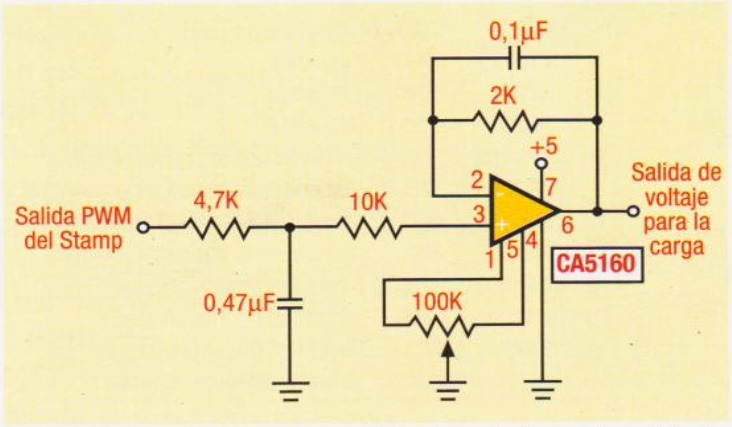


Figura 27.5 *Seguidor de voltaje de precisión con el amplificador operacional CA5160*

location: es una variable o una constante entre 0 y 255 en la que se especifica la posición de memoria que debe ser leída.
variable: es la variable de 8*bits* que recibe el dato leído de la posición de memoria especificada.

25. RETURN

Esta instrucción se usa para retornar el flujo del programa al cuerpo principal, al finalizar la ejecución de la subrutina de salto más inmediata (GOSUB).

26. REVERSE pin

Con esta instrucción el usuario puede invertir la configuración actual de un pin del puerto del microcontrolador en todo momento, si el pin es una entrada se convierte en una salida, y viceversa.

pin: es la variable o la constante entre 0 y 7 con la que se especifica el pin que se va a invertir.

27. SERIN pin,baudmode,(qualifier,qualifier,...) SERIN pin,baudmode,{#}variable, {#}variable,... SERIN pin,baudmode,(qualifier,qualifier,...), {#}variable,{#}variable,...

Por medio de esta instrucción, que se puede escribir en los tres formatos aquí indicados, se habilita un pin del puerto como una entrada serial y se entra en un estado de espera para las restricciones y las variables opcionales. Esta instrucción recibe un *byte* a la vez y lo compara con la o las restricciones, si ésta o éstas se satisfacen, se guarda en la variable correspondiente.

pin: es la variable o la constante entre 0 y 7 con la que se especifica el pin que va a ser usado como puerto de entrada serial.
baudmode: es una variable o una constante entre 0 y 7 con la que se define el modo de trabajo y la velocidad de recepción serial; en la **tabla 27.3** se muestran los ocho posibles modos de configuración de la entrada serial,

#	Símbolo	Velocidad	Polaridad
0	T2400	2.400	No invertida
1	T1200	1.200	No invertida
2	T600	600	No invertida
3	T300	300	No invertida
4	N2400	2.400	Invertida
5	N1200	1.200	Invertida
6	N600	600	Invertida
7	N300	300	Invertida

Tabla 27.3 *Tabla de opciones del modo de configuración de la entrada serial*

éstos pueden escribirse usando los números del 0 al 7, o los símbolos equivalentes para cada modo. En esta tabla aparecen los parámetros que se pueden cambiar, los otros son los usados por el formato serial estándar, denominado a menudo N81, y que son: sin *bit* de paridad, 8*bits* de datos y con un *bit* de parada.

qualifiers: son las variables o las constantes opcionales (entre 0 y 255) que representan las restricciones que se deben recibir en el orden especificado en el formato de la instrucción, antes de continuar con la ejecución del programa.

variables: son las variables opcionales que se usan para almacenar los datos recibidos (pueden ser hasta 255). Si están precedidas por el símbolo # la instrucción convierte el texto numérico en un valor antes de guardarlo en la variable.

Como un ejemplo, la siguiente instrucción detiene la ejecución del programa hasta que se reciben serialmente por P0, a 2.400*bauds* y con polaridad invertida, los *bytes* que conforman la cadena de caracteres CONTINUAR. Una vez que la restricción se cumple, se guarda el *byte* siguiente en la variable b3.

SERIN 0,N2400, (CONTINUAR),b3

28. SEROUT pin,baudmode,({#}data,{#}data,...)

Por medio de esta instrucción se habilita un pin del puerto del microcontrolador como una salida serial apta para la transmisión de datos.

pin: es la variable o la constante entre 0 y 7 con la que se especifica el pin que va a ser usado como puerto de salida serial.

baudmode: es una variable o una constante entre 0 y 15 con la que se define el modo de trabajo y la velocidad de transmisión serial; en la **tabla 27.4** se muestran los posibles modos de configuración de la salida serial, éstos pueden escribirse usando los números del 0 al 15, o los símbolos equivalentes para cada modo. En esta tabla aparecen los parámetros que se pueden cambiar, los otros son los usados por el formato serial estándar que se denomina **N81**.

data: son las variables o las constantes opcionales (entre 0 y 255) de 8*bits* cada una, que contienen el dato que se va a enviar serialmente. Cuando el dato está precedido por el símbolo # se transmite como una cadena de caracteres de texto (con longitud entre uno y cinco caracteres); pero cuando no, los datos se transmiten como un simple *byte*.

Como un ejemplo se escriben las dos siguientes instrucciones de transmisión serial a 1.200*bauds* y con polaridad invertida: con la primera se envía el valor 65 que equivale y se recibe como la letra "A" de acuerdo con el código ASCII; mientras que con la segunda se envía la cadena de texto de caracteres numéricos "65".

#	Símbolo	Velocidad	Polaridad y modo de salida
0	T2400	2.400	No invertida
1	T1200	1.200	No invertida
2	T600	600	No invertida
3	T300	300	No invertida
4	N2400	2.400	Invertida
5	N1200	1.200	Invertida
6	N600	600	Invertida
7	N300	300	Invertida
8	OT2400	2.400	No invertida (drenador abierto)
9	OT1200	1.200	No invertida (drenador abierto)
10	OT600	600	No invertida (drenador abierto)
11	OT300	300	No invertida (drenador abierto)
12	ON2400	2.400	Invertida (fuente abierta)
13	ON1200	1.200	Invertida (fuente abierta)
14	ON600	600	Invertida (fuente abierta)
15	ON300	300	Invertida (fuente abierta)

Tabla 27.4 *Tabla de opciones del modo de configuración de la salida serial*

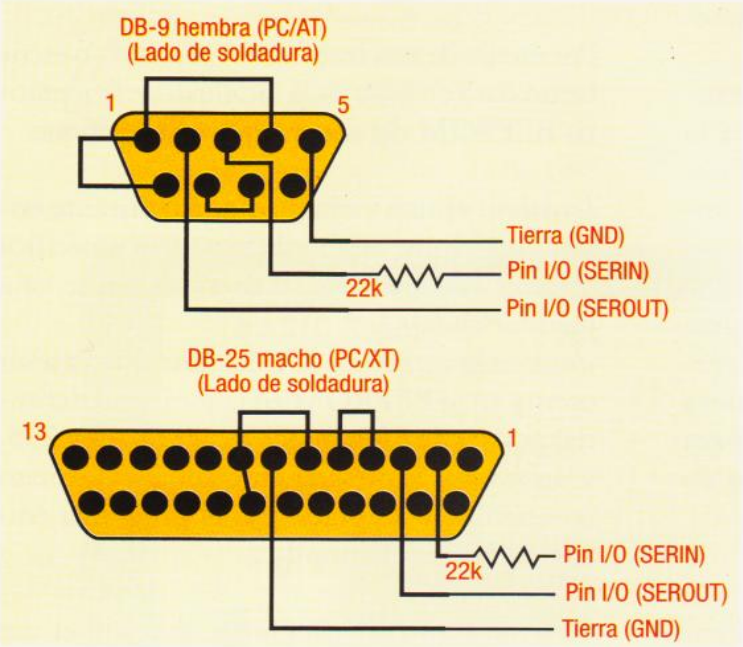


Figura 27.6 *Estructura de los conectores DB de 9 y 25 pines para transmitir datos entre la PC y el BASIC Stamp I*

SEROUT 1,N1200, (65)	'se recibe la letra "A"
SEROUT 1,N1200, (#65)	'se recibe la cade-'na de texto "65"

Para enviar o recibir datos seriales hacia y desde la PC, el BASIC Stamp I requiere de una resistencia de 22kΩ y alguna de las dos estructuras para los conectores de los puertos DB de 9 y 25 pines que se muestran en la **figura 27.6**.

29. SLEEP seconds

Esta instrucción hace que el microcontrolador entre en modo de adormecimiento o de bajo consumo (modo *sleep*) durante un número especificado de segundos. En este modo el consumo del microcontrolador se reduce de 1mA a cerca de 20µA.

seconds: es la variable o la constante en la que se especifica la duración del modo de adormecimiento en segundos, y puede estar entre 1 y 65.535 con una resolución de 2,304s; esto quiere decir que su duración puede estar entre 2,3s y 18h.

Si el microcontrolador está manejando cargas y entra en el modo *sleep*, es importante que el usuario siempre recuerde que la corriente a través de estas será interrumpida durante 18ms cada vez que éste despierta, debido a la ejecución del *reset* del temporizador del *watchdog*, lo cual se genera cada 2,304 segundos.

30. SOUND pin,(note,duration,note, duration,...)

Esta instrucción convierte el pin especificado en una salida y genera por él un tren de pulsos de onda cuadrada con una frecuencia determinada (una nota), y con una duración determinada. Para conectar directamente el parlante a la salida de tono, se pueden usar los dos circuitos que se muestran en la **figura 27.7**.

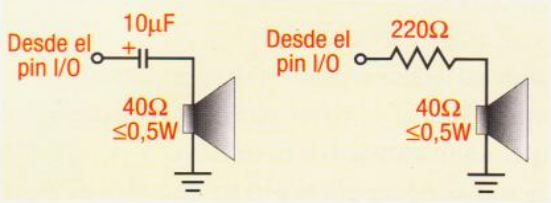


Figura 27.7 *Circuitos típicos para conectar directamente un parlante de alta impedancia al puerto del BASIC Stamp I*

pin: es la variable o la constante por medio de la cual se especifica el pin I/O que se va a usar como salida de sonido, puede estar entre 0 y 7.
note(s): son variables o constantes que pueden estar entre 0 y 255 y por medio de las cuales se define la frecuencia del tono de salida. Con la nota 0 se produce silencio (0Hz), con las notas 1 a 127 se producen tonos de frecuencia ascendente entre 94,8Hz y

$$\text{NOTA} = 127 - \frac{\dfrac{1}{\text{Frecuencia (Hz)}} - 95\text{X}10^{-6}}{83\text{X}10^{-6}}$$

$$\text{FRECUENCIA (Hz)} = \frac{1}{95 \text{ X } 10^{-6} + ((127 - \text{Nota}) \text{ X } 83 \text{ X } 10^{-6})}$$

Figura 27.8 *Ecuaciones para determinar la frecuencia y la nota del tono de salida*

10.550Hz, y con las notas 128 a 255 se generan ruidos blancos ascendentes.
duration(s): son las variables o las constantes por medio de las cuales se especifica la duración de la emisión de cada nota. Pueden estar entre 1 y 255, y la unidad de tiempo es 12ms.

En las ecuaciones de la **figura 27.8** se muestran las dos relaciones que hay entre la frecuencia de salida y la nota, cualesquiera de éstas puede ser de mucha utilidad para el usuario a la hora de calcular el tono de salida.

31. TOGGLE pin

Esta instrucción se utiliza para convertir el pin I/O especificado en una salida y cambiar su estado lógico de 1 a 0, y viceversa, cada que se ejecuta.

pin: es una variable o una constante que está entre 0 y 7 y sirve para especificar el pin al que se le cambiará el estado.
A manera de ejemplo se escribe el siguiente programa: en él se configura como salida el pin 7 y se cambia su estado lógico 10 veces.

```
FOR b1 = 1 TO 10
TOGGLE 7            'se cambia el estado de P7
NEXT
```

32. WRITE location,data

Por medio de esta instrucción se guarda o escribe un dato de 8*bits* en la localidad de la memoria EEPROM del sistema que se especifique.

location: es una variable o una constante entre 0 y 255 por medio de la cual se especifica la posición de la memoria en la cual se va a guardar el dato.
data: es la variable o la constante que va a ser escrita en la EEPROM en la localidad definida, por ser de 8*bits* puede estar entre 0 y 255, y debe estar expresada en cualquier formato compatible y aceptado por el programa editor del BASIC Stamp I.

Con el siguiente programa se aclara el uso de esta instrucción, con esta pequeña rutina se reciben datos en serie y se almacenan en posiciones consecutivas de la EEPROM.

```
READ 255,b2        'captura la posición de la última
                   'instrucción del programa
Repetir:
b2=b2-1            'decrementa hasta la próxi-
                   'ma posición de la EEPROM
                   'disponible
SERIN 0,N300,b3    'recibe un byte en serie en
                   'b3
WRITE b2,b3        'guarda el byte recibido en
                   'la siguiente posición de la
                   'EEPROM
IF b2>0 THEN Repetir 'salta a capturar otro byte
```

PARA SABER MÁS:
Redes de comunicación con BASIC STAMP I

En el diagrama que se muestra en la **figura 27.9** se indica, por medio de un circuito simple, la forma de usar la tecnología de drenador abierto (*open drain*), para conectar dos o más microcontroladores BASIC Stamp I con una PC, o cualquier terminal, a través de una línea de comunicación serial de una red de datos.

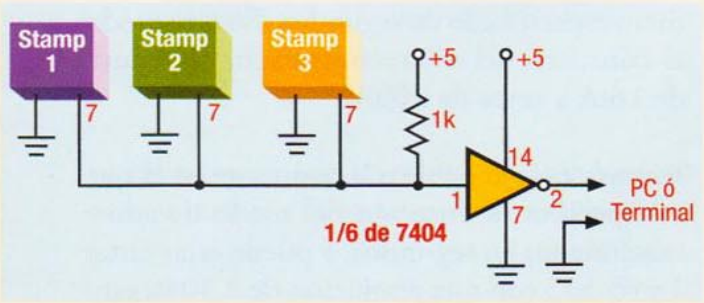
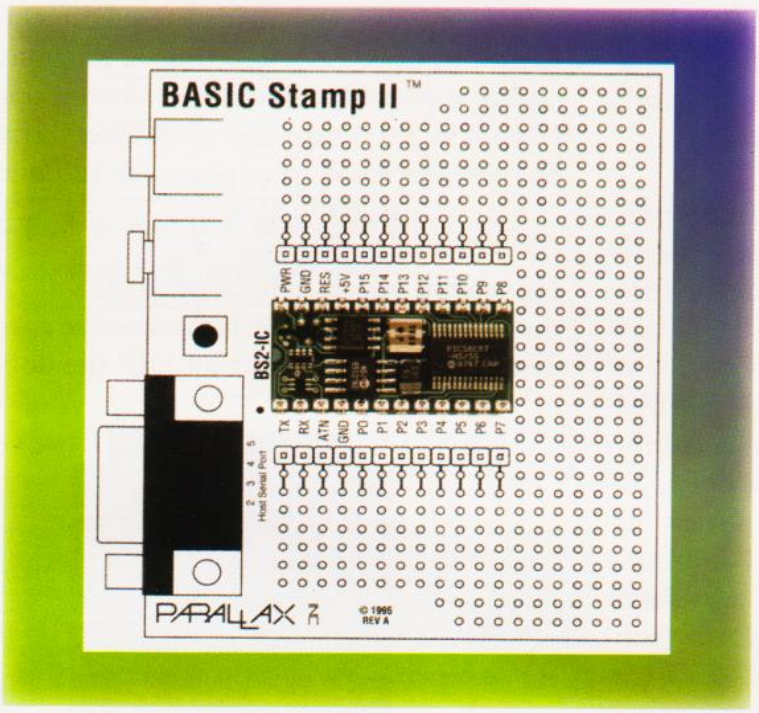


Figura 27.9 *Circuito para conectar múltiples sistemas BASIC Stamp I en red*



LOS MICROCONTROLADORES BASIC Stamp II

Generalmente los productos tecnológicos poseen un sucesor que es una nueva versión con otras funciones y características innovadoras, pero compatible 100% con sus antecesores; tal es el caso del microcontrolador BASIC Stamp II, que es la siguiente versión del BASIC Stamp I.

Para el BASIC Stamp II se ha desarrollado un lenguaje denominado PBASIC2, y se han agregado algunas herramientas de *hardware* y *software* de gran utilidad para el programador, además de una arquitectura y un soporte de programación mejorados.

El BASIC Stamp II

En esta lección estudiaremos el microcontrolador BASIC Stamp II (BS2-IC), desarrollado por Parallax, Inc. Este microcontrolador se construyó usando la tecnología de montaje superficial para obtener un dispositivo de bajo costo, reducido tamaño, alta eficiencia y gran capacidad, que puede ser usado como una pequeña computadora en trabajos para la industria, el comercio, la educación, el entretenimiento y el hogar.

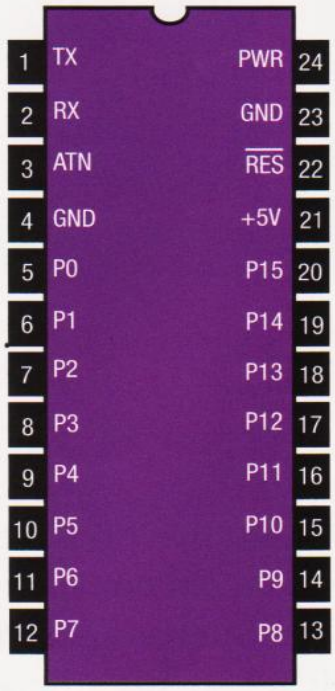
El BASIC Stamp II se compone de un entorno de programación amigable, un lenguaje de programación propio y una estructura de circuitos extendida como soporte. El lenguaje de programación se denomina PBASIC2 y ha sido desarrollado por Parallax a partir del lenguaje PBASIC con un reducido número de instrucciones (36 instrucciones), todas muy compactas, versátiles, poderosas y de fácil manejo; entre las nuevas instrucciones se pueden enunciar: COUNT (rutina para contar ciclos), RCTIME (rutina para medir las constantes de tiempo RC), FREQOUT (rutina para generar señales seno), y DTMFOUT (rutina para generar tonos telefónicos), entre otras.

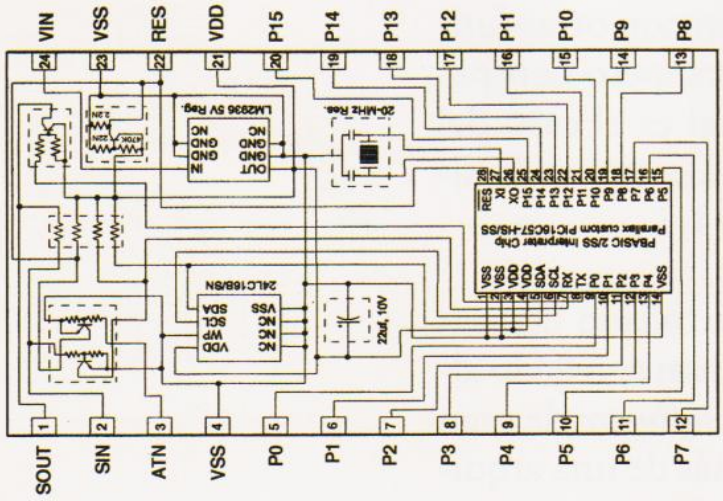
El *hardware* del microcontrolador BASIC Stamp II se construye sobre un pequeño circuito impreso de montaje superficial, y en el cual se incluyen: un microcontrolador PIC de Microchip con el programa residente (*firmware*) del intérprete de las instrucciones, una memoria de acceso serial tipo EEPROM, un regulador de +5V, un oscilador, un circuito de *reset* automático, y una interfaz de comunicación serial.

Para programar el BASIC Stamp II el usuario sólo debe contar con algunos requerimientos mínimos en su equipo, que son: el programa adecuado, una computadora con sistema operativo MS-DOS 2.0 ó superior, 128Kbytes disponibles en memoria RAM, y un puerto serial libre.

Distribución de pines del BASIC Stamp II

La versión del BASIC Stamp BS2-IC que se muestra en la figura 28.1, tal como se aprecia, se desarrolla en una estructura DIP (de doble fila)





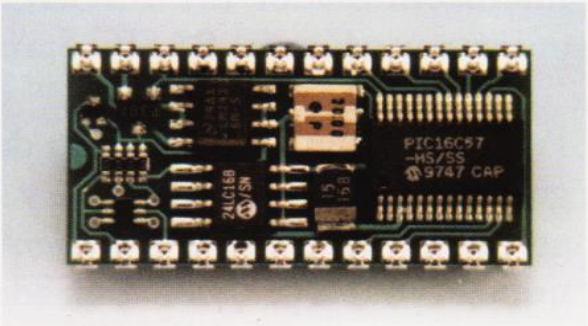


Figura 28.1 Aspecto físico y distribución de los componentes y pines del BASIC Stamp II

PIN	NOMBRE	DESCRIPCIÓN	COMENTARIOS
1	TX	Salida serial	Conectar al pin 2 del puerto serial DB9 (RX)
2	RX	Entrada serial	Conector al pin 3 del puerto serial DB9 (TX)
3	ATN	Atención, reset activo en alto	Conectar al pin 4 del puerto serial DB9 (DTR)
4	GND	Tierra serial	Conectar al pin 5 del puerto serial DB9 (GND)
5	P0	Pin I/O 0	Cada pin puede manejar 20 mA en modo source y 25 mA en modo sink P0-P7 y P8-P15 como grupos, pueden manejar un total de 40 mA en modo source y 50 mA en modo sink
6	P1	Pin I/O 1	
7	P2	Pin I/O 2	
8	P3	Pin I/O 3	
9	P4	Pin I/O 4	
10	P5	Pin I/O 5	
11	P6	Pin I/O 6	
12	P7	Pin I/O 7	
13	P8	Pin I/O 8	
14	P9	Pin I/O 9	
15	P10	Pin I/O 10	
16	P11	Pin I/O 11	
17	P12	Pin I/O 12	
18	P13	Pin I/O 13	
19	P14	Pin I/O 14	
20	P15	Pin I/O 15	
21	+5V	Fuente de +5V	Entrada regulada de + 5V y salida regulada
22	RES	Reset activo en bajo	Llevar a 0 para resetear (se pone en 0 durante el reset)
23	GND	Tierra del sistema	
24	PWR	Entrada al regulador	Entrada al regulador de voltaje de +5V (6-15 voltios)

Para selección automática del puerto serial, por parte del software del BASIC Stamp II,se deben unir los pines DSR (pin6) y RTS (pin 7) del puerto serial DB9. Esta conexión está hecha en la tarjeta de prototipos del BASIC Stamp II (carrier board), pero si se desea hacer una tarjeta especifica, se debe tener presente hacer dicha conexión.

Durante la operación normal el Stamp II consume cerca de 8mA, sin embargo en los modos de operación de bajo consumo se puede reducir a cerca de 50µA.

Tabla 28.1 *Distribución y descripción de pines del BASIC Stamp II*

de 24 pines para facilitar su acople y compatibilidad con los otros circuitos impresos y tableros de conexiones (*protoboards*). La asignación y las funciones definidas para los pines de este microcontrolador se describen a continuación en forma resumida en la **tabla 28.1**.

Arquitectura del BASIC Stamp II

En la figura 28.2 se muestra el diagrama electrónico del BASIC Stamp II, en ella pueden verse: un microcontrolador de 28 pines de Microchip PIC16C57 (U1) que es el "corazón" del sistema y en el cual está grabado el programa intérprete de las instrucciones del PBASIC2, una memoria serial EEPROM de 2.048bytes de Microchip 24LC16B (U2) que recibe y guarda el programa diseñado por el usuario, un circuito automático de *reset* a la conexión y desconexión (U3), un regulador de voltaje integrado de +5V (U4) que alimenta a todo el sistema, un circuito de oscilación de 20MHz, y una interfaz de comunicación serial RS232 para niveles de comunicación de +12V y -12V (Q1, Q2 y Q3).

La tarjeta de prototipos del BASIC Stamp II

En la figura 28.3 se muestra la tarjeta para desarrollo de prototipos (*carrier board*) diseñada por Parallax para el microcontrolador BASIC Stamp II, ésta cuenta con terminales para la conexión con una batería de 9VCC, un conector (*socket*) tipo DIP para la conexión del BASIC Stamp II, dos conectores en línea paralelos al microcontrolador que dejan disponibles todos sus pines, un pulsador para el *reset*, un conector BD9 hembra para la comunicación serial con la PC, y la tarjeta multiusos sobre la cual se pueden desarrollar pe-

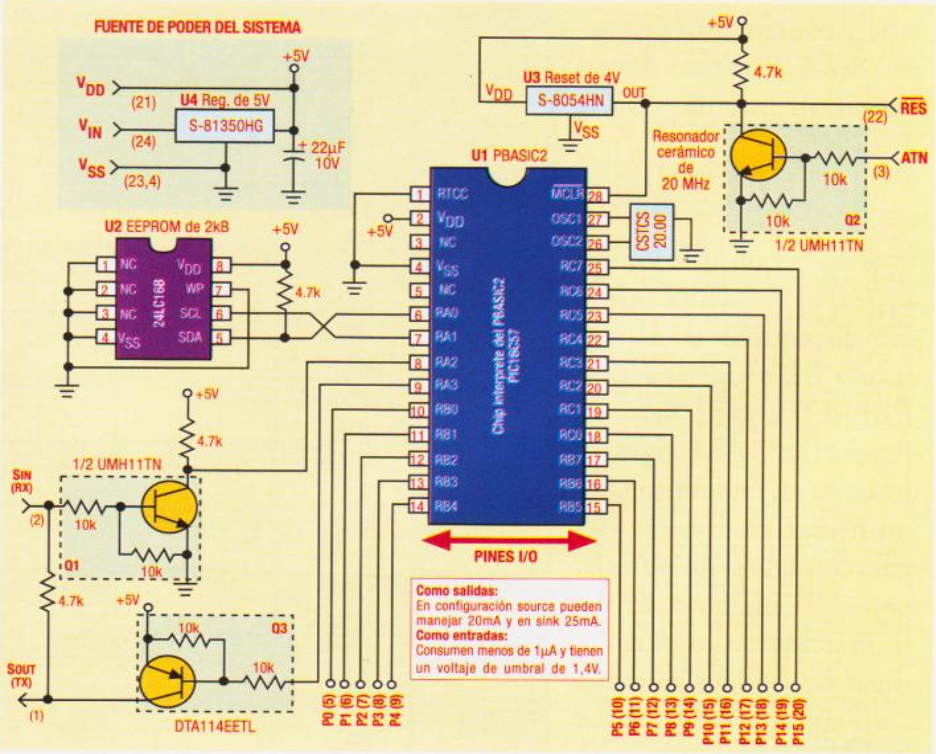


Figura 28.2 *Diagrama electrónico del BASIC Stamp II*

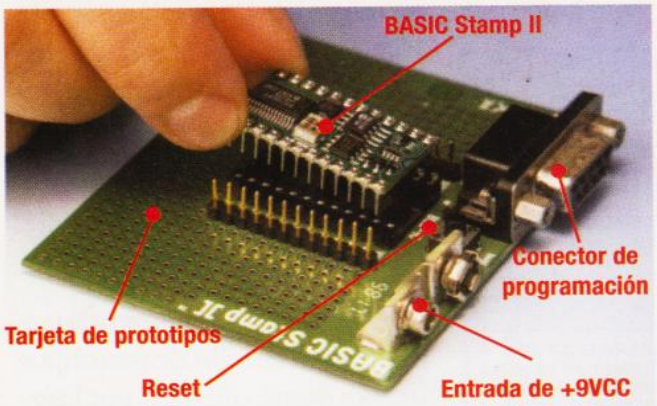


Figura 28.3 *Tarjeta de prototipos del microcontrolador BASIC Stamp II*

queños circuitos prototipo, lo que resulta bastante útil en la etapa de experimentación y desarrollo de cualquier proyecto.

Los circuitos que se construyan sobre la tarjeta de prototipos, pueden ser alimentados con el regulador interno de +5V de todo el sistema; pero, en vista de que el regulador tiene una capacidad máxima de 50mA, y el microcontrolador consume 8mA, éstos no deben consumir más de 42mA.

Mapa de memoria del BASIC Stamp II

Este microcontrolador tiene 2.048 posiciones de 8 *bits* disponibles en la memoria de programa tipo EEPROM y 32 *bytes* de memoria RAM para el usuario; en ésta última, los primeros 6 están reservados para configurar los pines de entrada o salida y controlar la dirección de los mismos, mientras que los restantes 26 están disponibles como variables de propósito general. En la figura 28.4 se describen las localidades del mapa de memoria RAM disponibles para el programador, y los registros, nombres, y funciones definidas para cada una.

Configuración de los puertos I/O del BASIC Stamp II

Las primeras tres posiciones del área de memoria son palabras de 16 *bits* (*words*) y se asocian con el control de los puertos; la primera palabra, la variable INS, es de sólo lectura y contiene el reflejo del estado lógico presente en los 16 pines I/O del puerto (P0 a P15), mientras que la segunda variable OUT, es de lectura y escritura, y contiene el estado binario de las salidas.

Mapa de memoria RAM del BASIC Stamp II

Palabra (16bits)	Byte (8bits)	Nibble (4bits)	Bit	Notas especiales
INS	INL INH	INA, INB, INC, IND	IN0 - IN7, IN8 - IN15	Pines de entrada; palabra, byte, nibble y bit direccionables
OUTS	OUTL OUTH	OUTA, OUTB, OUTC, OUTD	OUT0 - OUT7, OUT8 - OUT15	Pines de salida; palabra, byte, nibble y bit direccionables
DIRS	DIRL DIRH	DIRA, DIRB, DIRC, DIRD	DIR0 - DIR7, DIR8 - DIR15	Pines I/O de control de dirección; palabra, byte, nibble y bit direccionables
W0	B0 B1			De propósito general; palabra, byte, nibble y bit direccionables
W1	B2 B3			De propósito general; palabra, byte, nibble y bit direccionables
W2	B4 B5			De propósito general; palabra, byte, nibble y bit direccionables
W3	B6 B7			De propósito general; palabra, byte, nibble y bit direccionables
W4	B8 B9			De propósito general; palabra, byte, nibble y bit direccionables
W5	B10 B11			De propósito general; palabra, byte, nibble y bit direccionables
W6	B12 B13			De propósito general; palabra, byte, nibble y bit direccionables
W7	B14 B15			De propósito general; palabra, byte, nibble y bit direccionables
W8	B16 B17			De propósito general; palabra, byte, nibble y bit direccionables
W9	B18 B19			De propósito general; palabra, byte, nibble y bit direccionables
W10	B20 B21			De propósito general; palabra, byte, nibble y bit direccionables
W11	B22 B23			De propósito general; palabra, byte, nibble y bit direccionables
W12	B24 B25			De propósito general; palabra, byte, nibble y bit direccionables

Figura 28.4 *Mapa de la memoria RAM disponible del BASIC Stamp II*

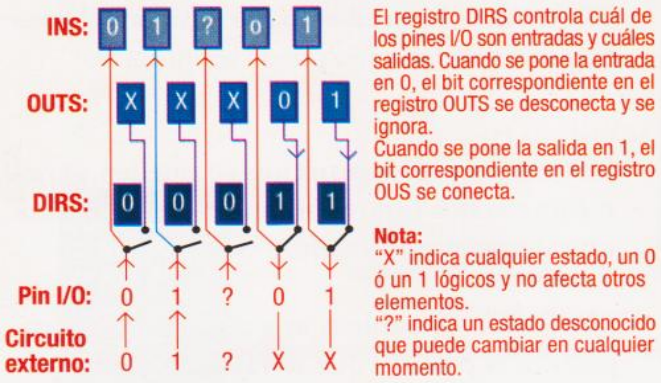


Figura 28.5 *Diagrama resumido de interacciones entre las variables DIRS, INS y OUTS*

Con la palabra DIRS se configura la dirección (como entrada o salida) de cada uno de los 16 pines del puerto.

En la **figura 28.5**, por medio de un diagrama, se resumen los resultados de las relaciones y combinaciones de los estados lógicos entre las variables de control DIRS, OUTS e INS.

Programación del BASIC Stamp II

En lo que a la programación de este microcontrolador se refiere, además de conocer las instrucciones del lenguaje PBASIC2, el usuario debe conocer el programa de desarrollo que suministra el fabricante, desde el cual, se puede editar, compilar, depurar y transferir el programa. Para programar y depurar el *software* del microcontrolador se utiliza el puerto serial de la computadora y el cable DB-9 a DB-9 que el fabricante suministra especialmente para este propósito, o uno similar; en la **figura 28.6** se muestra la configuración que tiene el conector que se incluye en la tarjeta de prototipos del BASIC Stamp II.

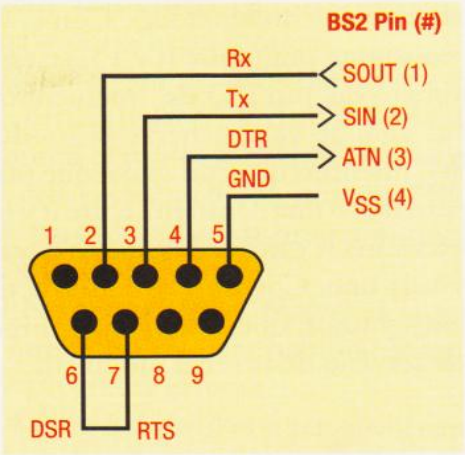


Figura 28.6 *Configuración del conector de comunicaciones del BASIC Stamp II*

Instrucciones del lenguaje PBASIC2

El lenguaje de programación del BASIC Stamp II está compuesto por un conjunto reducido de instrucciones (36 instrucciones) de alto nivel y se denomina PBASIC2; éste ha sido diseñado por Parallax teniendo en cuenta las instrucciones de la versión anterior (PBASIC) y agregando algunas otras con funciones especiales.

En esta sección se estudiarán las nuevas instrucciones de este lenguaje y las modificaciones que se han hecho sobre las de la versión anterior (PBASIC), estudiadas antes. Se presentarán en orden alfabético para facilitar su búsqueda y omitiendo las instrucciones que existen y operan igual en las dos versiones del lenguaje, así como también, aquellas que sólo sufren una extensión en la variable o constante llamada pin, que en el Stamp II puede estar entre 0 y 15 y que se refiere al número del pin sobre el cual se está trabajando.

1. COUNT pin,period,variable

Con esta instrucción se cuenta el número de ciclos (cambios de estado de 0 a 1 a 0, ó, de 1 a 0 a 1) que lleguen por un pin especificado durante un periodo de tiempo definido en milisegundos y se guarda el valor de la cuenta en una variable. Para ondas rectangulares el contador puede responder hasta una frecuencia de 125kHz.

pin: es una variable o una constante que especifica el pin I/O que se va a usar como entrada del contador, puede estar entre 0 y 15.
period: es una variable o una constante entre 0 y 65.535 con la cual se especifica el tiempo (en ms) que va a durar la cuenta de los ciclos.
variable: es la variable, generalmente una palabra, en la cual se debe guardar el resultado del contador de ciclos.

2. DEBUG outputdata{,outputdata...}

Con esta instrucción se puede monitorear el curso del programa en cualquier momento y por medio de ella se puede visualizar cualquier variable o mensaje en la pantalla de una PC conectada con el BASIC Stamp II.

outputdata: puede ser una o varias de las siguientes opciones: cadenas de texto, variables, constantes, expresiones, modificadores de formato, o caracteres de control.

3. DTMFOUT pin,{ontime,offtime,}{,tone...}

Utilizando esta instrucción se pueden generar tonos duales de múltiples frecuencias (*Dual Tone MultiFrecuency*, DTMF), compatibles con el sistema de comunicación por tonos de las redes telefónicas.

pin: es una variable o una constante que especifica el pin I/O que se va a usar como salida, está entre 0 y 15.
ontime: es una variable o una constante opcional entre 0 y 65.535 con la cual se especifica el tiempo (en ms) que va a durar el tono. Por defecto, si no se especifica, es 200ms.
offtime: es una variable o una constante opcional entre 0 y 65.535 con la cual se especifica el tiempo (en ms) que va a durar el espacio de silencio entre los tonos, o al final del tono. Por defecto, si no se especifica es igual a 50ms.
tone: es una variable o una constante (entre 0 y 15) que especifica el tono DTMF que debe ser enviado. Entre 0 y 9 corresponde con los tonos de los dígitos 0 a 9, con 10 equivale al carácter star (*), con 11 al carácter número (#), y entre 12 y 15 corresponde con la cuarta columna de tonos, A a D, usados en equipos de prueba telefónicos y de radio.

4. FOR variable = start TO end {STEP stepval}...NEXT

Con esta instrucción se implanta un lazo del tipo FOR...NEXT tal como en la instrucción de la versión anterior, pero, en este caso, el incremento se ha cambiado por el valor del paso (*stepval*) que también puede ser negativo.

5. FREQOUT pin,duration,freq1{,freq2}

Utilizando esta instrucción se pueden generar uno o dos tonos con ondas seno durante un tiempo determinado.

pin: es una variable o una constante que especifica el pin I/O que se va a usar como salida, está entre 0 y 15.
duration: es una variable o una constante entre 1 y 65.535 con la cual se especifica el tiempo (en ms) que van a durar los tonos.
freq1 y freq2: son las variables o las constantes, la segunda opcional, por medio de las cuales se especifica la frecuencia del tono en Hertz; puede estar entre 0 y 32.767.

6. PULSOUT pin,time

Por medio de esta instrucción se puede generar un pulso por uno de los pines del puerto (0 al 15) durante un tiempo que puede estar entre 2ms y 131ms, con resolución de 2ms. La variable o constante *time* puede estar entre 1 y 65.535.

7. PWM pin,duty,cycles

Usando esta instrucción se puede producir una señal modulada por ancho de pulsos (PWM) en cualquiera de los pines I/O del puerto, y convertir un valor digital en uno análogo (un voltaje entre 0 y 5V desarrollado sobre los terminales de un condensador). Con la variable o la constante pin (entre 0 y 15) se especifica el número del pin I/O del microcontrolador por el cual se va a generar la modulación PWM, mientras que con *cycles*, que puede ser una variable o una constante entre 0 y 65.535, se determina la cantidad de ciclos de salida de 1ms cada uno. Con *duty* se define la relación de ceros y unos, que es conocida como el ciclo de servicio de la señal modulada.

8. RCTIME pin,state,resultvariable

Utilizando esta instrucción se cuenta el tiempo durante el cual un determinado pin permanece

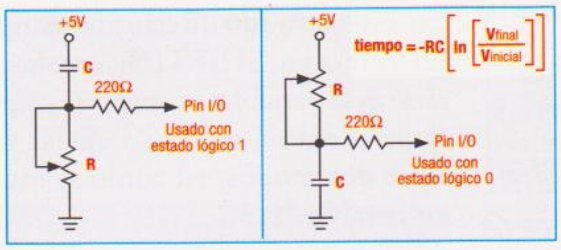


Figura 28.7 *Redes sugeridas para medir la constante de tiempo RC y la ecuación característica*

en un estado. Esta instrucción se ha diseñado para medir los tiempos de carga y descarga de condensadores en redes con resistencias (RC).

pin: es una variable o una constante que especifica el pin I/O que se va a usar como entrada, está entre 0 y 15.
state: es una variable o una constante que puede ser 1 ó 0, y, por medio de ella, se define el estado que se debe alcanzar para finalizar la medición del periodo RC.
resultvariable: es la variable en la cual se guarda el tiempo que ha resultado de la medida, el alcance está entre 0 y 65.535 en unidades de 2s.

En la **figura 28.7** se muestran dos tipos de redes que pueden ser utilizadas para medir el periodo RC, y la ecuación que se usa para ello.

9. SERIN rpin{\fpin},baudmode,{plabel,} {timeout,tlabel,}[inputdata]

Por medio de esta instrucción se habilita un pin del puerto como una entrada serial de datos no sincrónica (tal como la RS-232).
rpin: es la variable o la constante entre 0 y 16 con la que se especifica el pin que va a ser usado como puerto de entrada serial. Con 16 se especifica la entrada serial dedicada del sistema (SIN).
fpin: es una variable o una constante opcional (entre 0 y 15) con la que se define cuál va a ser el pin de control de flujo de datos.
baudmode: es la variable o constante de 16 *bits* con la que se define el tiempo y la configuración de datos. Los *bits* del 0 al 13 definen el tiempo para un *bit* recibido menos 20s, el *bit* 13 controla la longitud del dato y la paridad (0= 8 *bits* sin paridad, y 1= 7 *bits* con paridad par), y el *bit* 14 controla la polaridad (0= directa, y 1=invertida).
plabel: es la etiqueta opcional con la que se nombra la posición a la que se debe saltar si se detecta un error de paridad.
timeout: es una variable o constante opcional (de 0 a 65.535) que sirve para definir el tiempo (en ms) que se debe esperar para recibir un dato.
tlabel: es una etiqueta opcional que hace referencia a la posición a la cual se debe saltar si se detecta que un dato no ha entrado en el tiempo especificado por *timeout*.
inputdata: es una lista de variables y modificadores que le informan al microcontrolador lo que se debe hacer con los datos que van llegando en serie.

10. SEROUT tpin,baudmode,{pace,}[outputdata] SEROUT tpin\fpin,baudmode,{timeout,tlabel,} [outputdata]

Por medio de los dos formatos de esta instrucción, que aquí se muestran, se habilita un pin del puerto como una salida serial de datos no sincrónica (tal como la RS232).

tpin: es la variable o la constante entre 0 y 16 con la que se especifica el pin que va a ser usado como un puerto de salida serial. Con 16 se elige la salida serial dedicada del sistema (SOUT).
baudmode: es la variable o constante de 16*bits* con la que se define el tiempo y la configuración de datos. La función asignada para sus *bits* es similar a la de la instrucción SERIN.
pace: es una variable o constante opcional (entre 0 y 65.535) que sirve para definir el tiempo (en ms) que se debe esperar para transmitir un nuevo dato.
outputdata: es una lista de variables, constantes y modificadores que le informan al microcontrolador cuál es el formato de los datos que se van a enviar.
fpin: es la variable o la constante opcional entre 0 y 15 con la que se define cuál va a ser el pin para el control del flujo de datos durante la rutina de transmisión *byte* a *byte* (*handshaking*).

SÍMBOLO	VALOR	SIGNIFICADO
MSBPRE	0	Primero el MSB y muestreo antes de los pulsos del reloj
LSBPRE	1	Primero el LSB y muestreo antes de los pulsos del reloj
MSBPOST	2	Primero el MSB y muestreo después de los pulsos del reloj
LSBPOST	3	Primero el LSB y muestreo después de los pulsos del reloj

Tabla 28.2 *Modos de recepción sincrónica con la instrucción SHIFTIN*

timeout: es una variable o constante opcional (de 0 a 65.535) que se usa para definir el tiempo (en ms) que se debe esperar para recibir el permiso para enviar un dato.
tlabel: es una etiqueta opcional que hace referencia a la posición a la cual se debe saltar, al detectar que no se ha dado el permiso de transmisión dentro del tiempo definido en *timeout*.

11. SHIFTIN dpin,cpin,mode,[result{\bits},result {\bits}...]

Con esta instrucción se reciben datos en formato serie y con desplazamiento sincrónico.

dpin: es la variable o la constante entre 0 y 16 con la que se especifica el pin receptor del dispositivo de transmisión serial sincrónica.
cpin: es la variable o la constante entre 0 y 16 con la que se especifica el pin que va a ser usado como el reloj de sincronización.
mode: hay cuatro modos que pueden definirse con sus símbolos o con los números entre 0 y 3. En la **tabla 28.2** se describen dichos modos
result: puede ser una variable tipo *bit, nibble, byte,* ó *word*. En ésta se guardan los *bits* del dato.
bits: es una entrada opcional que define el número de *bits* que deben ser recibidos (entre 1 y 16). Si no se especifica, se asume como 8.

12. SHIFTOUT dpin,cpin,mode,[data{\bits},data {\bits}...]

Con esta instrucción se envían datos en formato serie y con desplazamiento sincrónico.

SÍMBOLO	VALOR	SIGNIFICADO
LSBFIRST	0	El primer dato desplazado por la salida es el LSB
MSBFIRST	1	El primer dato desplazado por la salida es el MSB

Tabla 28.3 *Modos de transmisión sincrónica con la instrucción SHIFTOUT*

En este caso los interruptores operan tal como en SHIFTIN, excepto que *data* es la variable o constante que contiene el dato que se va a enviar y sólo tiene dos modos; tal como se muestra en la **tabla 28.3**.

13. STOP

Con esta instrucción se detiene la ejecución en curso del programa (se "congela"), pero el microcontrolador no reduce el consumo de potencia.

14. WRITE address,byte

Con esta instrucción se escribe directamente un *byte* en la memoria EEPROM del sistema.

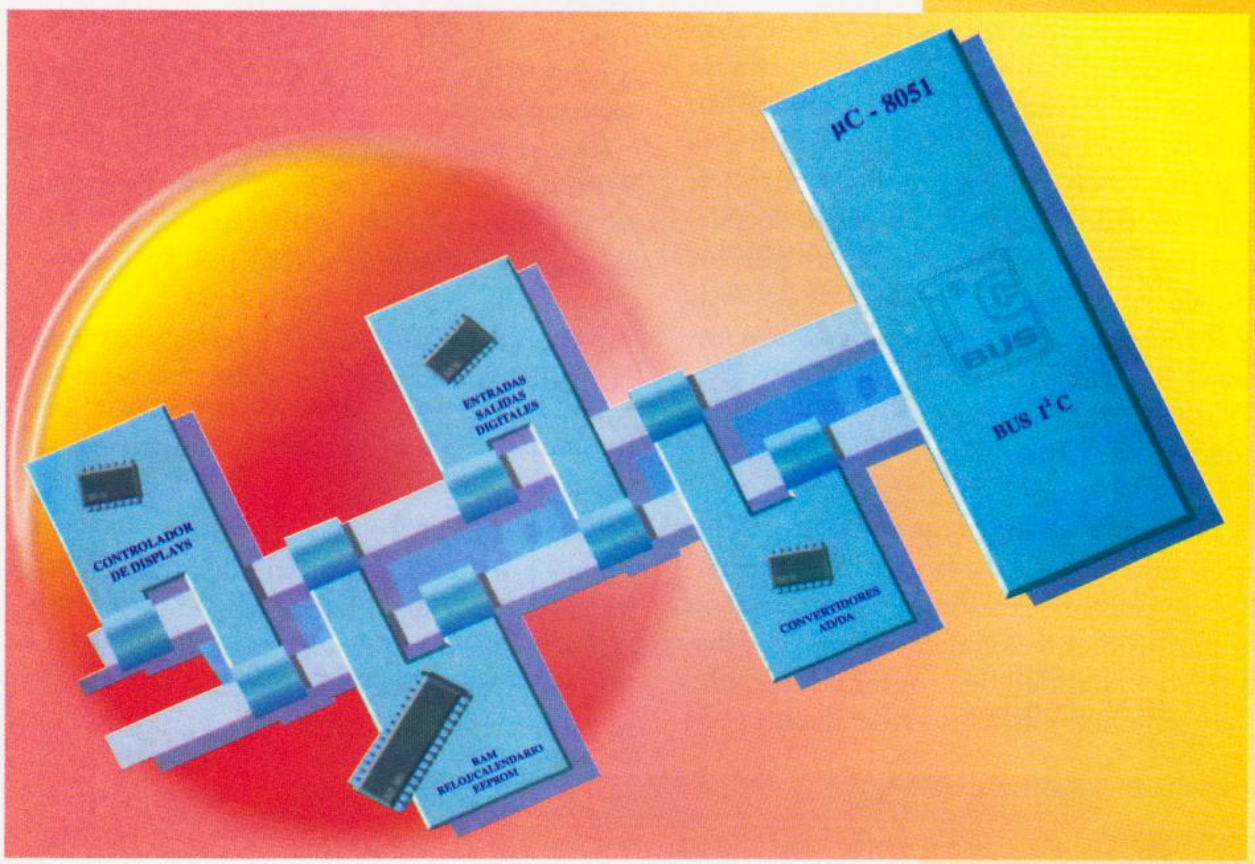
address: es la variable o la constante con la que se especifica la localidad de la memoria que va a ser escrita con un dato de 8*bits*. Puede estar entre las posiciones 0 y 2.047.
byte: es el dato con formato de un *byte* que se va a guardar en la memoria.

15. XOUTmpin,zpin,[house\keyorcommand {\cycles},...]

Con esta instrucción se pueden enviar hasta 10 comandos a través de un pin del puerto, el cual es usado como línea de control. Éstos se envían modulados y deben recibirse por medio de la interfaz apropiada.

mpin: especifica el pin I/O del puerto (entre 0 y 15) que se usará como salida múltiple de comandos (todos modulados).
zpin: especifica el pin I/O (entre 0 y 15) que se usará como entrada de cruce por cero desde la interfaz.
house: especifica el código de los comandos enviados, con valores entre 0 y 15 se representan los comandos equivalentes para las letras entre la A y la P.
keyorcommand: especifica la tecla o el comando de control equivalente.
cycles: es un interruptor opcional y por defecto permanece en 2. Se usa para especificar el número de veces que se debe transmitir el comando.

LECCIÓN 29



LOS MICROCONTROLADORES 8051 de Intel

Los microcontroladores desarrollados por Intel, han sido los pioneros en el manejo y desarrollo de tareas que involucran sistemas de control embebidos de pequeña y mediana envergadura.

Dentro de la amplia gama de éstos, la familia MCS-51, que se empezará a estudiar en esta lección, es la generación avanzada de 8*bits* que posee una arquitectura optimizada para la ejecución de las tareas orientadas al control digital de procesos en tiempo real.

Los microcontroladores Intel

Intel ha sido una de las empresas pioneras en el desarrollo de microcontroladores compatibles 100% con la arquitectura de las PC.

Estos microcontroladores han evolucionado a través de las familias MCS-48 (primera generación de 8*bits*), MCS-51(generación avanzada de 8*bits*), y MCS-96 (generación de 16*bits* y alta velocidad de procesamiento); y, también, han sido fabricados en versiones particulares por muchas otras empresas que los distribuyen y utilizan en sus propios dispositivos y sistemas, tal es el caso de los microcontroladores PHILIPS serie 87CXX, de los Dallas DS5000 y los Siemens SAB 8051, todos equivalentes a los Intel 8051.

La familia MCS-51, cuyo estudio se empezará en esta lección, es la que más popularidad tiene por sus excelentes características de compatibilidad, velocidad, bajo costo, disponibilidad y facilidad de programación; además, hay muchos fabricantes de sistemas de desarrollo y entrenadores para esta tecnología que nos facilitan su aprendizaje.

Entre los sistemas de desarrollo más conocidos se pueden nombrar: el **EV80C51** de Intel, el **CEIBO DS-750** de Philips Semiconductors y el **DS 5000TK** de Dallas Semiconductors; también se ofrecen otras herramientas, como el emulador avanzado **ICE-5100** de Dallas y circuitos para entrenamiento, tal como el **entrendor para 87C752** con referencia **K- 144** de **CEKIT S.A.**.

En la **tabla 29.1** se muestran las características principales de los microcontroladores de la familia MCS-51.

Arquitectura básica de los microcontroladores 8051

En la **figura 29.1** se muestra la arquitectura interna básica de los microcontroladores 8051, en ella se destacan las principales partes que los conforman. Dentro de las características generales se destacan las siguientes:

	8051AH	8751H	80C51BH	87C51	8052AH	8752BH
ROM	4KB	4KB EPROM	4KB	4KB EPROM	8KB	8KB EPROM
RAM	128	128	128	128	256	256
XROM	64KB	64KB	64KB	64KB	64KB	64KB
XRAM	64KB	64KB	64KB	64KB	64KB	64KB
FRECUENCIA (MHz)	12	12	16	12	12	12
CICLO	1	1	75	75	1	1
TIMERS	2	2	2	2	3	3
SERIAL	UART	UART	UART	UART	UART	UART
I/O	32	32	32	32	32	32
INTERRUPCIONES	5	5	5	5	6	6
IccMax a 12MHz	125	250	20	25	175	175
IDLE Icc (TYP: mA a 12MHz)	--	--	1,7	1,3	--	--
IPD Icc (TYP: µA)	--	--	5	3	--	--

Tabla 29.1 *Características resumidas de los microcontroladores de la familia MCS-51*

- CPU de 8*bits*
- Procesador booleano (con operación lógica sobre *bits*)
- 4 puertos de 8*bits* cada uno
- 128*bytes* de RAM interna para datos
- 4K*bytes* de memoria ROM ó de EPROM para el programa
- 64K*bytes* de espacio direccionable externamente para datos o programa
- 2 temporizadores-contadores (*timer* 0 y *timer* 1)
- Comunicación no sincrónica *full-duplex*
- 5 fuentes de interrupción
- Oscilador interno

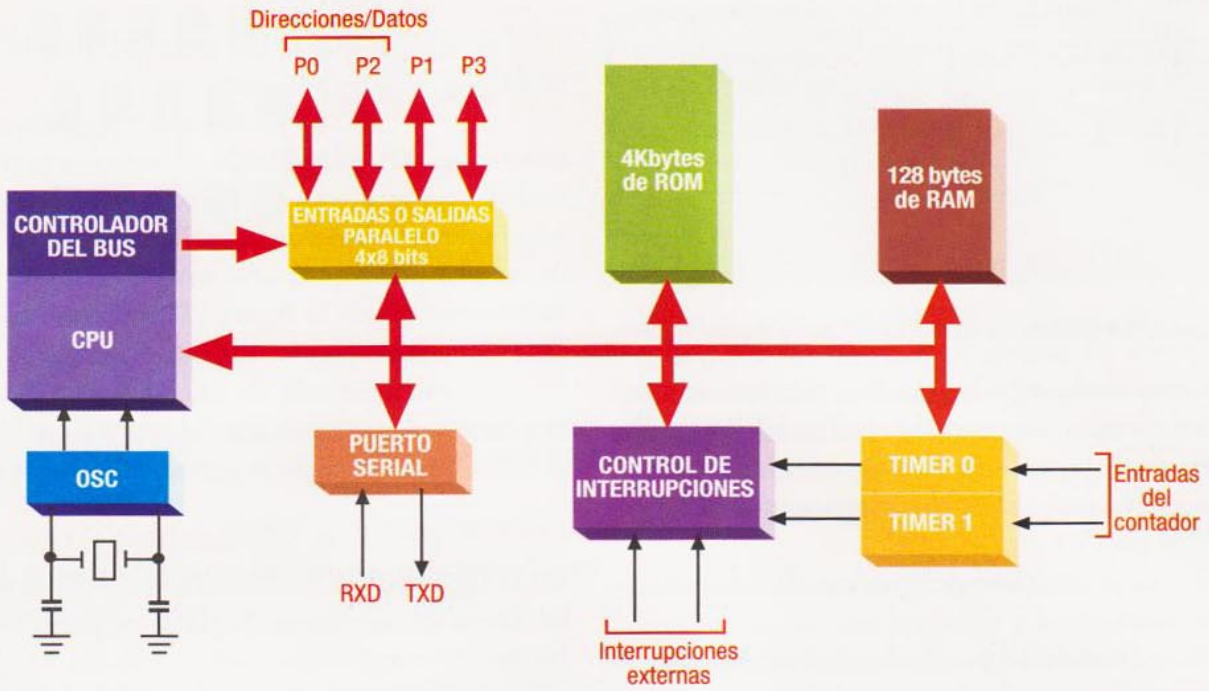


Figura 29.1 *Arquitectura interna de los microcontroladores 8051*

Descripción de pines del microcontrolador 8051

En la **figura 29.2** se muestra la distribución de los pines del microcontrolador 8051 en su formato de empaque tipo DIP. A continuación se hará una descripción breve de todos sus pines y sus funciones.

Vcc: es la entrada de alimentación regulada positiva de +5V

Vss: es la tierra del sistema, 0V.

P0, P1, P2 y P3: son cuatro puertos de 8*bits* bidireccionales. Todos tienen una capacidad de carga (*fan-out*) de 4 cargas TTL-LS, exceptuando el P0 que soporta hasta 8 cargas.

ALE/PROG: ALE (*address latch enable*) se emite con una frecuencia igual a 1/6 de la del reloj y se usa para capturar el *byte* bajo del bus de direcciones durante el acceso a memoria externa. PROG es la entrada para los pulsos de programación de la memoria EPROM interna.

PSEN: PSEN (*program store enable*) es la señal de control que se utiliza para habilitar y leer la memoria de programa externa.

EA/VPP: EA (*external access*) es el pin que controla el modo de trabajo del dispositivo; cuando es igual a 1, sólo se ejecuta el programa desde la ROM interna (opera como microcontrolador); y cuando es

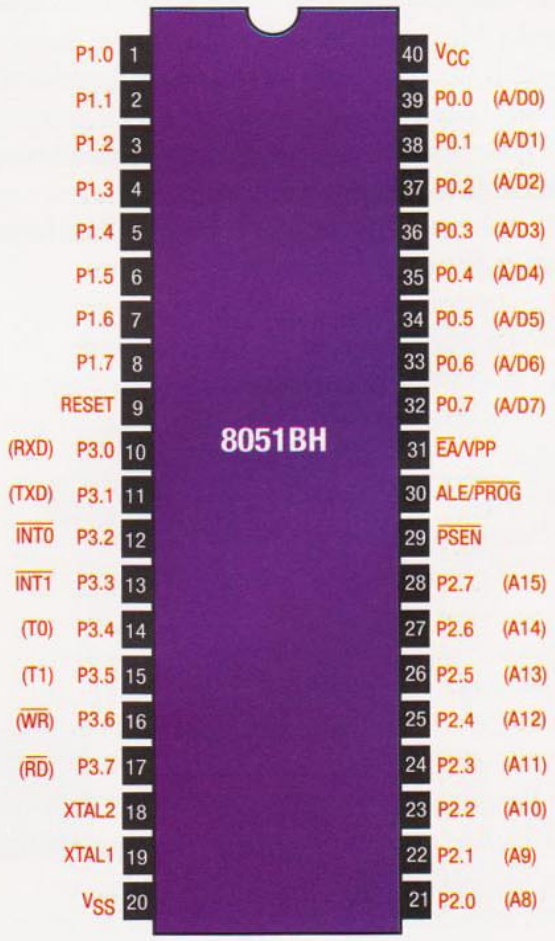


Figura 29.2 *Asignación de pines en los microcontroladores 8051*

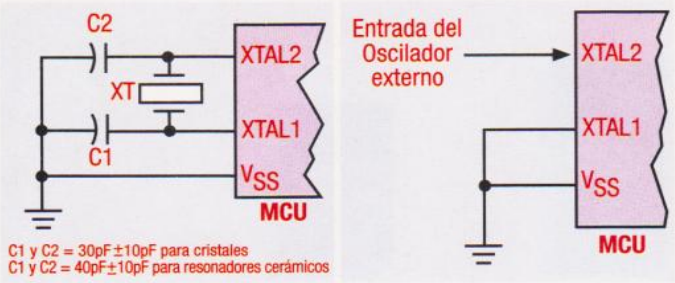


Figura 29.3 *Circuitos típicos para el oscilador interno y externo*

0, se ejecuta el programa desde la memoria externa (opera como microprocesador). Por VPP se recibe la tensión de programación necesaria para "quemar" el programa en la EPROM (típicamente está entre 12,75V y 21V).

RXD: es la entrada del puerto de comunicación serial.

TXD: es la salida del puerto de comunicación serial.

INT0: es la entrada de la señal de interrupción externa 0.

INT1: es la entrada de la señal de interrupción externa 1.

T0: es la entrada externa del temporizador-contador 0 (*timer* 0).

T1: es la entrada externa del temporizador-contador 1 (*timer* 1).

WR: es la señal que se usa para habilitar la escritura en dispositivos externos.

RD: es la señal que se usa para habilitar la lectura de dispositivos externos.

XTAL1 y XTAL2: son la entrada y la salida respectivamente, del amplificador inversor interno del oscilador, éste puede configurarse para generar el reloj internamente o para usarse como un circuito de acoplamiento para una señal de reloj generada externamente. En la **figura 29.3** se muestran dos circuitos típicos para configurar el oscilador.

RESET: es la entrada de inicialización del sistema. Su estado activo es alto (al ponerla en 1 lógico durante cierto tiempo se genera el ciclo de *reset*).

Organización de la memoria en el 8051

En la **figura 29.4** se muestra la distribución de las áreas de memoria de datos y programa de los microcontroladores de la familia MCS-51; aquí puede verse que estos microcontroladores tienen espacios de direcciones separados para las memorias de programa y de datos. La memoria de programa sólo puede ser leída, y puede ser direccionada externamente hasta 64K*bytes*, o internamente hasta 4K*bytes*; mientras que la memoria de datos puede ser leída o escrita, y también puede ser interna o externa con capacidad de direccionamiento hasta 256*bytes* y 64K*bytes*, respectivamente.

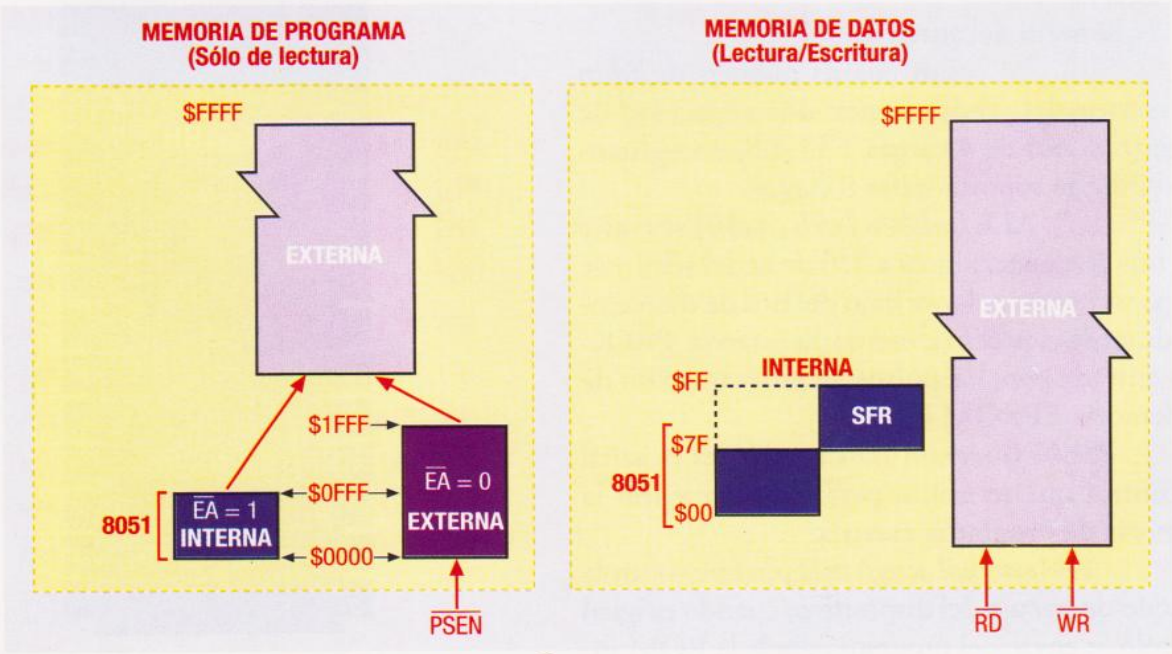


Figura 29.4 *Mapa de memoria del microcontrolador 8051*

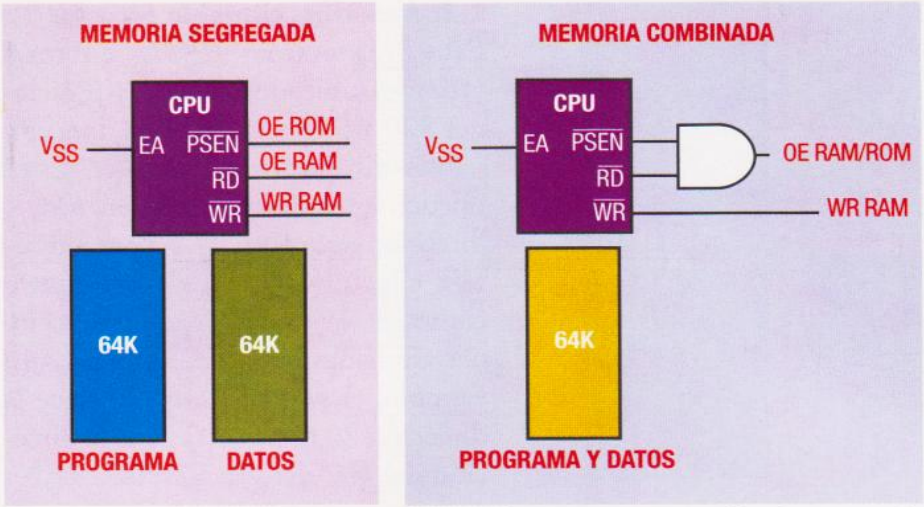


Figura 29.5 *Hardware para la configuración de áreas de memoria combinada y segregada*

La memoria de programa y la de datos pueden coexistir en el mismo espacio de direcciones cuando es **combinada**, o en espacios diferentes cuando es **segregada**; la memoria combinada se puede configurar por medio de una compuerta lógica AND conectada con las salidas RD y PSEN, las cuales son generadas por la CPU durante el acceso a la memoria externa. Por medio de la salida de dicha compuerta se pueden habilitar las salidas de datos del *chip* de memoria externa, tal como se aprecia en la **figura 29.5**.

Memoria de programa

La memoria de programa puede ser dividida, y de acuerdo con el programador, ser usada como una memoria mixta o externa; la selección se lleva a cabo por medio de la señal de control EA, poniendo el pin a +5V ó a 0V respectivamente. Cuando es mixta, se divide el espacio direccionable de 64K*bytes* en 4K*bytes* de memoria interna y 60K*bytes* de memoria externa, y, se debe tener presente, que en la parte baja de la memoria interna se encuentran las posiciones asignadas para los vectores de interrupción; tal como el vector del *reset* que ocupa la posición $0000. Cuando es externa, el usuario define todo el mapa de la memoria de programa dentro de los 64K*bytes* accesibles externamente; en la **figura 29.6** se muestran las dos configuraciones posibles de la memoria de programa.

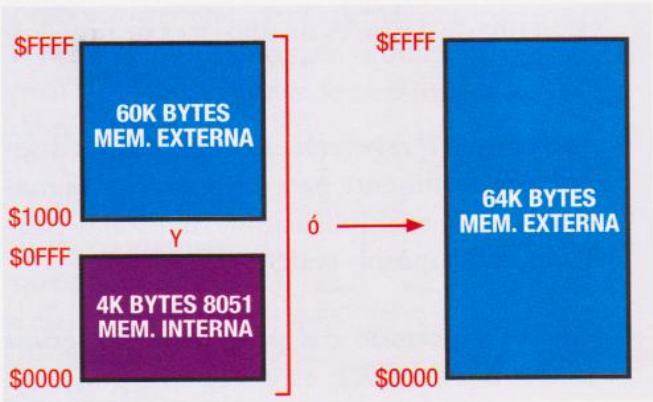


Figura 29.6 *Configuración de la memoria de programa*

Memoria de datos

La memoria de datos se puede usar como una memoria interna o externa con capacidad de direccionamiento de 256*bytes* ó 64K*bytes*, respectivamente; y, para acceder a los datos de la memoria externa se utiliza la instrucción **MOVX**. La memoria de datos, cuando es interna, se compone de dos áreas que son: los 128*bytes* bajos ocupados por los **registros de propósito general** y los 128*bytes* altos en donde se encuentra el espacio reservado para los **registros de función especial** (*special function register*, SFR); en la **figura 29.7** se muestra el mapa de la memoria de datos.

1. Registros de propósito general

En la figura 29.7 se muestra que el direccionamiento de la memoria interna puede ser directo ó indirecto. El bloque de los 128*bytes* más bajos puede accederse en los dos modos de direccionamiento, y posee algunas características y asignaciones especiales, tales como son: los bancos de registros, el área direccionable *bit* a *bit* y el área de anotaciones de acceso rápido (*scratch pad*). En la **figura 29.8** se muestra la segmentación de la memoria de datos interna, y, a continuación, se estudiarán sus características.

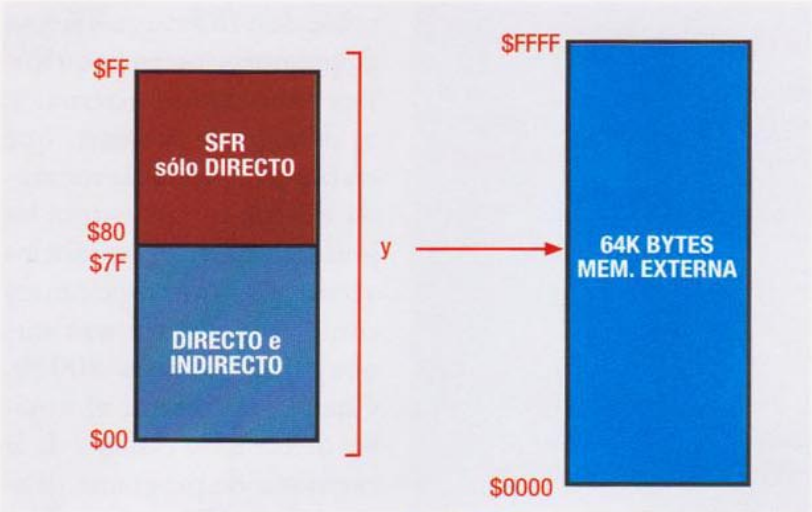


Figura 29.7 *Mapa de la memoria de datos*

Figura 29.8 *Segmentación de la memoria de datos interna*

1.1. Bancos de registros

Los bancos de registros son cuatro (banco 0 a banco 3) y cada uno se compone de 8 registros de 8*bits* (R0 a R7); éstos se encuentran entre las posiciones $00 y $1F y ocupan 32*bytes*. El banco 0 es el seleccionado por defecto luego de un *reset*, mientras que la selección de los otros bancos debe hacerse por *software* desde el registro de estado PSW (*program status word*), que hace parte de los SFR.

Nota:
Por defecto, el *reset* hace que el registro puntero de la pila (SP) señale a la posición que corresponde con el registro R0 del banco 2 ($08). Si se quieren usar más bancos de registros, el SP se puede inicializar después del *reset* en otra posición de memoria.

1.2. Área direccionable *bit* a *bit*

Esta área ocupa 128*bits* en total (16*bytes*) ubicados entre las posiciones $20 y $2F, y todos ellos direccionables directamente *bit* a *bit*. Éstos pueden además agruparse en *bytes* y direccionarse como tal, y, para referirse a ellos directamente como *bits* o como *bytes*, se deben usar las direcciones asignadas para cada uno; así, por ejemplo, el *bit*0 del *byte* $20 tiene la dirección 20.0, mientras que la dirección del *bit*7 del *byte* $2F es la 2F.7.

1.3. Área de anotaciones (*scratch pad*)

Esta área de acceso directo e indirecto es la memoria RAM de trabajo del programador, es un bloque de memoria de acceso rápido pero de baja capacidad, y en este microcontrolador se encuentra entre las localidades de memoria $30 y $7F.

2. Los registros de función especial SFR

En el bloque de los 128*bytes* más altos de la memoria de datos se encuentran los registros de función especial (SFR) de esta familia de microcontroladores, y en la **tabla 29.2** se muestran los SFR del MCU 8051, en ella se nombran y se definen sus símbolos y direcciones. A continuación se explican brevemente las funciones de los SFR, y los más importantes de ellos se verán en detalle en su debido momento.

Registro acumulador (***Accumulator register***) **(ACC ó A)**: es el más utilizado durante el movimiento de datos, es de 8*bits* y es de propósito general.

Registro B (***B register***): este registro está diseñado especialmente para las operaciones matemáticas de multiplicación y división, y puede ser de propósito general.

Palabra de estado del programa ***(Program Status Word*** **PSW)**: es el registro que contiene la información del estado de la CPU,

SÍMBOLO	NOMBRE		DIR
*ACC	Accumulator	Acumulador	$E0
*B	B register	Registro B	$F0
*PSW	Program status word	Palabra de estado del programa	$D0
SP	Stack pointer	Puntero de la memoria de pila	$81
DPTR	Data pointer (2 bytes)	Puntero de datos	--
DPL	Low byte data pointer	Byte bajo del puntero de datos	$82
DPH	High byte data pointer	Byte alto del puntero de datos	$83
*P0	Port 0	Puerto 0	$80
*P1	Port 1	Puerto 1	$90
*P2	Port 2	Puerto 2	$A0
*P3	Port 3	Puerto 3	$B0
*IP	Interrupt priority control	Control de prioridad de las interrupciones	$B8
*IE	Interrupt enable control	Control de autorización de las interrupciones	$A8
TMOD	Timer/Counter mode control	Control de modo del temporizador/contador	$89
*TCON	Timer/Counter control	Control del temporizador/contador	$88
*+T2CON	Timer/Counter 2 control	Control del temporizador/contador 2	$C8
TH0	Timer/Counter 0 high byte	Byte alto del temporizador/contador 0	$8C
TL0	Timer/Counter 0 low byte	Byte bajo del temporizador/contador 0	$8A
TH1	Timer/Counter 1 high byte	Byte alto del temporizador/contador 1	$8D
TL1	Timer/Counter 1 low byte	Byte bajo del temporizador/contador 1	$8B
+TH2	Timer/Counter 2 high byte	Byte alto del temporizador/contador 2	$CD
+TL2	Timer/Counter 2 low byte	Byte bajo del temporizador/contador 2	$CC
+RCAP2H	T/C 2 capture reg. high byte	Byte alto del registro de captura T/C 2	$CB
+RCAP2L	T/C 2 capture reg. low byte	Byte bajo del registro de captura T/C 2	$CA
*SCON	Serial control	Control de comunicaciones seriales	$98
SBUF	Seria data buffer	Buffer de datos de comunicaciones seriales	$99
PCON	Power control	Control del modo de consumo de potencia	$87

+ = Registros que se utilizan sólo en el MCU 8052
* = Direccionable bit a bit

Tabla 29.2 *Registros de función especial del MCU 8051*

y se actualiza con cada ciclo de instrucción. En el **cuadro 29.1** se muestra su formato.

Registro apuntador de la pila (*Stack pointer*): es el registro apuntador del área de la pila (que es de 256 posiciones), es de 8*bits* y señala a la posición $07 cada vez que un ciclo de *reset* es activado. Luego de ejecutar la primera instrucción de apilamiento o llamado (*PUSH* o *CALL*), queda señalando a la primera posición disponible de la pila ($08). En la **figura 29.9** se muestra el área de la pila.

Apuntador o puntero de datos (*Data pointer*): es el registro que contiene en todo momento la dirección del puntero de datos, puede ser de 16 ó de 8*bits* (DPTR, DPH ó DPL).

Puertos 0, 1, 2, 3 (*Ports* **P0, P1, P2, P3**): son los cuatro *latchs* de los puertos y son de 8*bits* cada uno.

Buffer **de datos seriales** (*Serial data buffer* **SBUF**): es el registro que opera como impulsador de datos de recepción o transmisión durante la comunicación serial.

Registros temporizadores (*Timer register* **TH/L0, TH/L1**): son registros de 16*bits* que operan como contadores o temporizadores y se componen a su vez de una parte baja (L) y otra alta (H), ambas de 8*bits*.

Registro de control de potencia (*Power control register* **PCON**): es el registro que controla el modo de consumo de potencia. En la versión del microcontrolador 8051 con tecnología CHMOS se encuentran disponibles dos modos de trabajo de bajo consumo: el modo *POWER DOWN* y el modo **IDLE**. Adicionalmente, se usa para controlar la velocidad de comunicación por el canal serial.

Registro de control del temporizador/contador (*Timer/Counter control register* **TCON**): es el registro que controla el funcionamiento de las interrupciones y los flancos de activación asignados para los *trimer* 0 y 1.

Figura 29.9 *La pila del MCU 8051*

PALABRA DE ESTADO DEL PROGRAMA PSW							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C	AC	F0	RS_1	RS_0	OV	--	P

BITS	NOMBRES Y COMENTARIOS
Bit0	**P** Bandera de paridad del acumulador (ACC) • Si P = 1 entonces el número de unos del ACC es impar • Si P = 0 entonces el número de unos del ACC es par
Bit1	Bandera disponible y definible por el usuario
Bit2	**OV** Bandera de sobrepasamiento (overflow)
Bit3 - Bit4	**RS_0 - RS_1** Selectores de los bancos de registros
Bit5	**F0** Bandera 0 de propósito general, definible por el usuario
Bit6	**AC** Bandera de acarreo auxiliar, para operaciones en BCD
Bit7	**C** Bandera de acarreo

RS_1	RS_0	BANCOS
0	0	Banco 0 ($00-$07)
0	1	Banco 1 ($08-$0F)
1	0	Banco 2 ($10-$17)
1	1	Banco 3 ($18-$1F)

Cuadro 29.1 *Formato del registro de estado del MCU 8051*

Registro de control de modo del temporizador/contador (*Timer/Counter mode control register* TMOD): es el registro con el cual se selecciona el modo de operación del *timer* 0 ó 1 (como temporizador ó como contador).

Registro habilitador de direcciones (*Interrupt enable register* IE): es el registro dedicado al control de las interrupciones; le permite al programador habilitar únicamente las interrupciones que necesite.

Registro de prioridad de interrupciones (*Interrupt priority register* IP): es el registro que controla la prioridad de cada fuente de interrupción habilitada, al poner en 1 ó en 0 sus *bits*. Cada fuente de interrupción podrá ser programada en nivel de prioridad 1 ó 2, y toda interrupción con prioridad superior puede bloquear la ejecución de otra con prioridad inferior, y no lo contrario.

Registro de control del puerto serial (*Serial port control register* (SCON): por medio de este registro se definen los parámetros de la comunicación serial, tales como la velocidad, el formato de la palabra y la dirección de la transmisión.

El *reset* en los MCU 8051

En el MCU 8051 con encapsulado DIP, el pin 9 es el *reset*; este pin se activa cuando recibe un nivel lógico alto (+5V) durante al menos dos ciclos de máquina (24 periodos del reloj del oscilador). Durante tal estado, la CPU escribe los parámetros iniciales en todos los registros que dependan de éste, e inicializa el *hardware* de todos los módulos periféricos internos; en la **tabla 29.3** se muestran los registros que son afectados por el ciclo de *reset*.

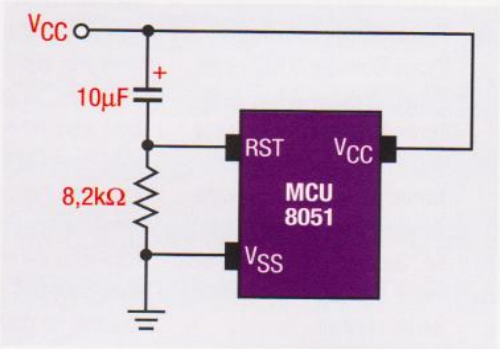


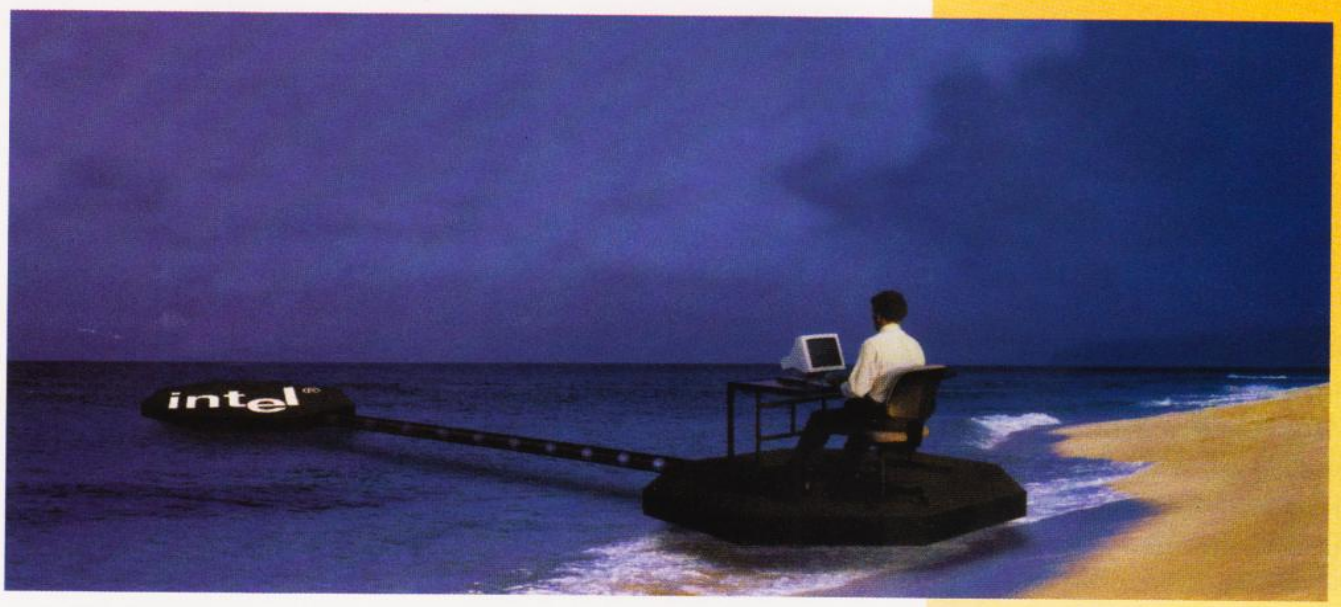
Figura 20.10 *Circuito típico de reset automático para el 8051*

El circuito que se muestra en la **figura 29.10** es útil para generar un reset automático a la conexión (*power on reset*), éste se produce al poner en el pin de *RESET* el voltaje de alimentación (Vcc) durante el tiempo definido por la red de carga RC, que está formada típicamente con una resistencia de 10kΩ y un condensador de10µF. Estos valores pueden ajustarse de acuerdo con el reloj aplicado al microcontrolador.

SFR	VALOR DESPUÉS DEL RESET
PC	$0000
ACC	$00
B	$00
PSW	$00
SP	$07
DPTR	$0000
P0-P3	$FF
IP (8051)	XXX00000B
IP (8052)	XX000000B
IE (8051)	0XX00000B
IE (8052)	0X000000B
TMOD	$00
TCON	$00
TH0	$00
TL0	$00
TH1	$00
TL1	$00
TH2 (8052)	$00
TL2 (8052)	$00
RCAP2H (8052)	$00
RCAP2L (8052)	$00
SCON	$00
SBUF	Indeterminado
PCON (HMOS)	0XXXXXXXB
PCON (CHMOS)	0XXX0000B

Tabla 29.3 *Estado inicial de los registros después del reset*

LECCIÓN 30



PUERTOS Y PERIFÉRICOS INTERNOS DEL MICROCONTROLADOR 8051

Los microcontroladores 8051 poseen puertos multifuncionales, éstos pueden operar como *drivers* para manejar directa o indirectamente cargas, u operar como buses bidireccionales de datos y direcciones durante el acceso a dispositivos externos.

Los módulos temporizadores y los contadores son los periféricos internos de mayor importancia, ya que se constituyen en la herramienta más útil con que cuenta el programador para los proyectos con el 8051.

PARA SABER MÁS:
El ciclo de máquina del 8051

En esta familia de microcontroladores, un **ciclo de máquina** que es la unidad básica de tiempo en la que se cuenta la duración de la ejecución de una instrucción u operación, consta de una secuencia de seis **estados** que se nombran entre S1 y S6; y, a su vez, cada estado se compone de dos periodos de la señal de reloj del oscilador llamados fases (fase 1, **P1** y fase 2, **P2**); tal como se puede ver en la **figura 30.1.** De lo anterior se deduce que cada ciclo de máquina tiene 12 fases y que su duración se puede determinar por medio de esta expresión: **CM = 12/Frecuencia**; esto quiere decir que, por ejemplo, para un reloj de 12MHz se obtendrá un ciclo de máquina de 1ms de duración.

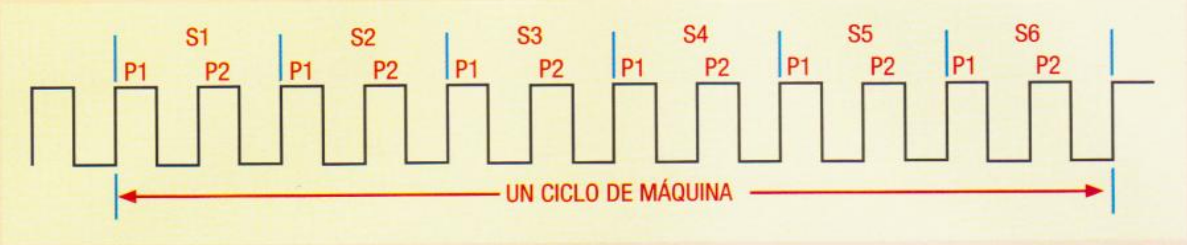


Figura 30.1 *Representación en el tiempo de un ciclo de máquina del 8051*

Los puertos del MCU 8051

En la **figura 30.2** se muestra la arquitectura interna de los puertos multifuncionales de esta familia de microcontroladores. El 8051 posee 4 puertos bidireccionales (de entrada o salida) e independientes, de 8*bits* cada uno y que se denominan P0, P1, P2 y P3. Como salidas, el *fan-out* del P0 es de ocho cargas TTL-LS, mientras que los otros puertos P1 a P3 solo pueden manejar cuatro cargas.

Para programar los puertos como salidas, es necesario escribir un **0** lógico en el *latch* correspondiente, mientras que si se escribe un **1** lógico se programan como entradas. Por su arquitectura *totem-pole*, solamente el P0 puede adoptar el estado de alta impedancia o tercer estado (*tristate*) cuando se programa como entrada, los otros asumirán un estado alto debido a la resistencia *pull-up* integrada en su estructura de drenador abierto (*open-drain*); en cualquier caso, para leer un dato el *latch* correspondiente al puerto debe cargarse con **1** lógico para que el transistor MOS inferior permanezca en estado de circuito abierto, y la resistencia de *pull-up* sea la impedancia vista por la entrada.

Las funciones compartidas y las características particulares de cada uno de los puertos se estudian a continuación:

1. Puerto 0 (P0): opera como un puerto de 8*bits*, pero también multiplexa en el tiempo la parte baja del bus de direcciones y todo el bus de datos durante el acceso a la memoria externa de programa o datos. Este puerto también se usa como bus de datos durante la programación y verificación de la memoria interna EPROM en algunas versiones del microcontrolador.

2. Puerto 1 (P1): opera como un puerto de 8*bits*, pero también recibe la parte baja del bus de direcciones durante la programación y verificación de la EPROM interna, en la versión 8751 de esta familia de microcontroladores.

3. Puerto 2 (P2): opera como un puerto de 8*bits*, pero también se usa como la parte alta del bus de direcciones durante el acceso a la memoria externa de programa o datos. También recibe

PINES	FUNCIÓN ALTERNA
P3.0	RXD (Entrada del puerto serial)
P3.1	TXD (Salida del puerto serial)
P3.2	$\overline{INT0}$ (Interrupción externa 0)
P3.3	$\overline{INT1}$ (Interrupción externa 1)
P3.4	T0 (Entrada externa del *Timer* 0)
P3.5	T1 (Entrada externa del *Timer* 1)
P3.6	$\overline{WR}$ (Autorización de escritura en memoria de datos externa)
P3.7	$\overline{RD}$ (Autorización de lectura en memoria de datos externa)

Tabla 30.1 *Funciones alternas del puerto 3*

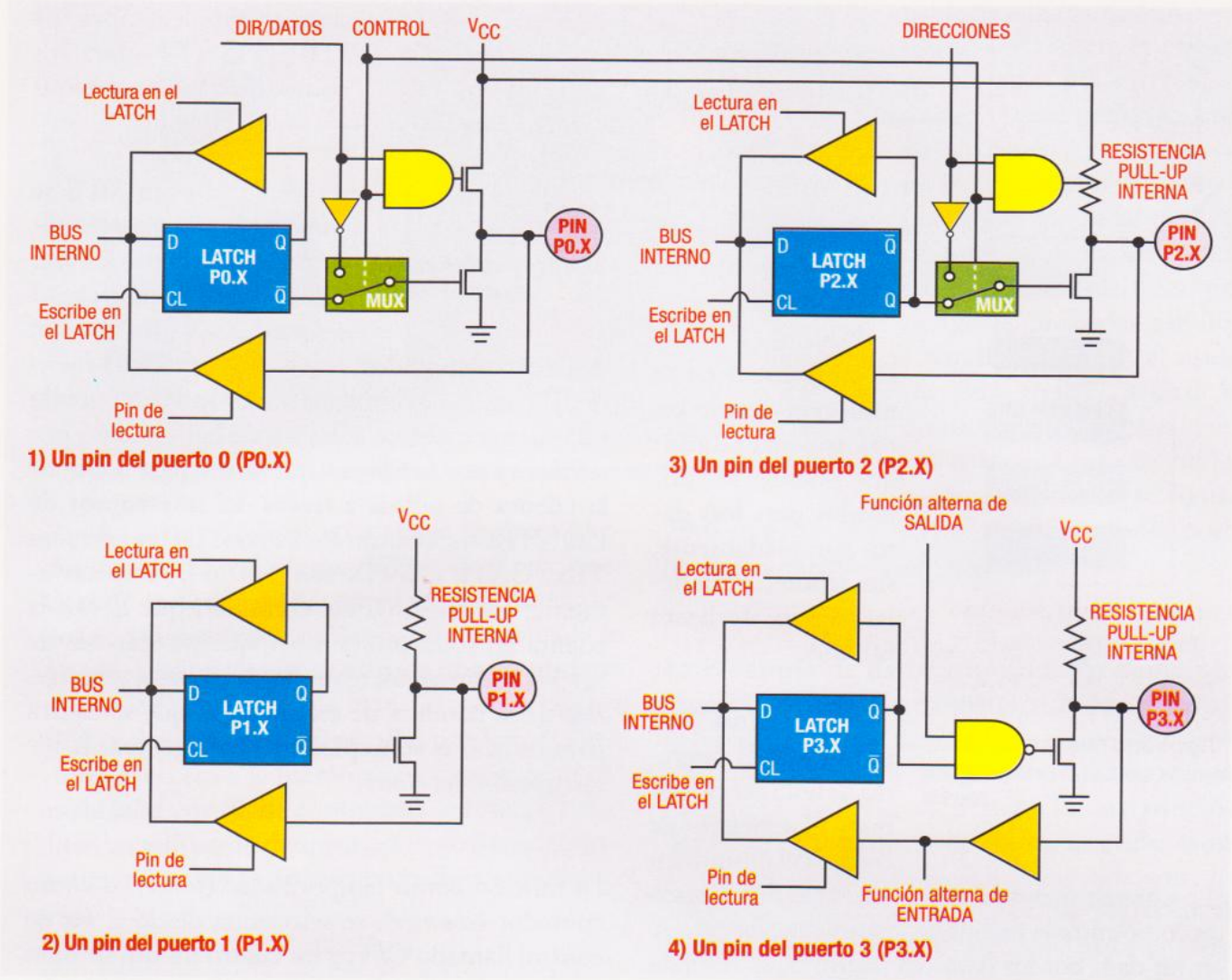


Figura 30.2 *Arquitectura de los puertos del MCU 8051*

la parte alta de la dirección durante la programación y verificación de los datos de la memoria EPROM.

4. Puerto 3 (P3): opera como un puerto de 8*bits* de propósito general, pero además comparte otras funciones importantes; en la **tabla 30.1** se resumen las otras funciones de cada uno de los pines de este puerto.

La operación de escritura en un puerto, que generalmente se efectúa por medio de la instrucción MOV Px,(DATO), se puede apreciar en el diagrama de tiempos de la **figura 30.3**, ésta se usa habitualmente para cargar en el puerto Px el DATO por medio de cualquier modo de direccionamiento.

Para la lectura sólo basta con invertir el orden de los operandos de la instrucción usada anteriormente para la escritura, así: MOV (DATO),Px.

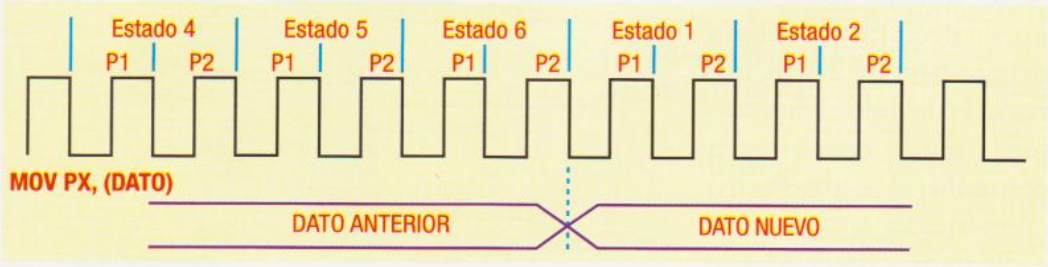
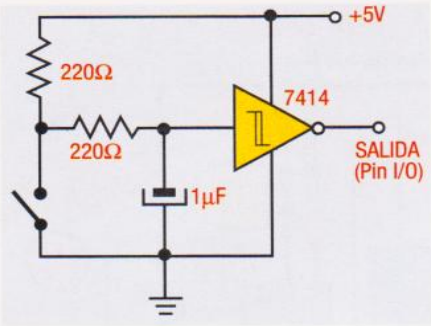


Figura 30.3 *Diagrama de tiempos de la instrucción MOV PX, (DATO)*



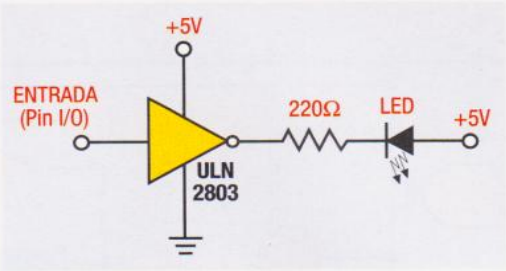


Figura 30.4 *Interfaces para leer y escribir en los puertos del 8051*

Figura 30.5 *Diagrama de flujo del programa ejemplo, para el uso de los puertos*

En la **figura 30.4** se muestran dos circuitos recomendados que pueden ser utilizados para leer datos apropiadamente, sin rebote, y manejar cargas de hasta 600mA.

En la **figura 30.5** se muestra el diagrama de flujo del programa y el segmento de código del mismo. En el programa se ilustra el uso adecuado de los puertos como entrada y salida de manera sencilla; en él se lee un dato por los 8*bits* del puerto 0, se complementa su estado y se envía por los pines del puerto 1.

Los temporizadores-contadores (timers 0 y 1) del MCU 8051

Esta familia de microcontroladores posee dos temporizadores-contadores (*timer* 0 y *timer* 1). Cuando opera como temporizador, el contador se incrementa con cada ciclo de máquina hasta sobrepasar el valor prefijado; por este motivo, se puede decir que la cuenta se hace con 1/12 de la frecuencia del reloj; mientras que cuando opera como **contador**, el registro es incrementado en uno con cada flanco descendente de la señal externa que se detecte por los pines 14 (T0) y 15 (T1) para los contadores de los *timers* 0 y 1, respectivamente.

En la **figura 30.6** se muestra la arquitectura general de los *timers* del 8051, allí pueden verse: el selector (**C/$\overline{T}$**) de la señal de reloj del contador, con el cual se elige la fuente del reloj a partir del oscilador interno o desde una fuente externa; la lógica de habilitación por *software* y por *hardware*, que se usa para habilitar la cuenta de pulsos a través del interruptor de CONTROL, cuando los estados de las entradas TRx, GATE e $\overline{INT}$x son 1, 0 ó 1, respectivamente; los registros del contador, que llevan la cuenta en todo momento y que pueden ser de 8,13 ó 16*bits* según el modo de trabajo; y, el *flip-flop* de la bandera de estado TFx, que se utiliza para indicar el sobrepasamiento y generar la interrupción del *timer*.

Modos de operación de los *timers* 0 y 1

La función como temporizador (*timer*), o como contador (*counter*), se selecciona desde el *bit* de control llamado **C/$\overline{T}$**; y los cuatro modos de operación de estos dos temporizadores-contadores se definen con los *bits* **M0** y **M1**. Todos estos tres *bits*, pertenecen al registro de control de modo **TMOD** (*Timer/counter Mode Control Register*) que hace parte de los **SFR**. En las **figuras 30.7** y **30.8** se muestran los registros de control y de modo **TMOD** y **TCON** (*timer/counter control register*), asociados con el comportamiento de los dos *timers*.

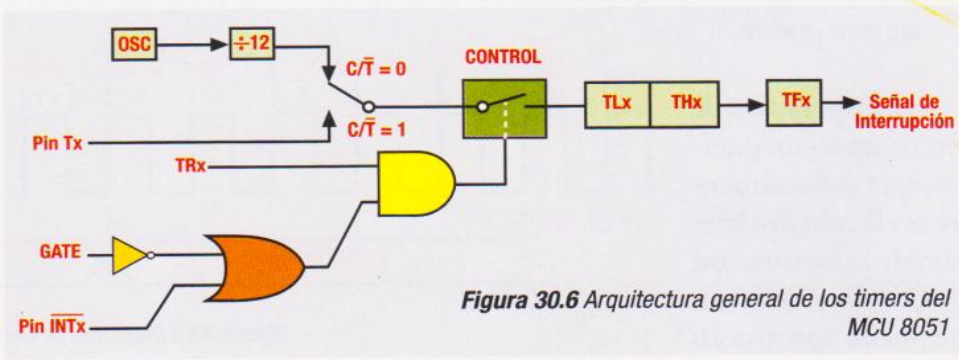


Figura 30.6 *Arquitectura general de los timers del MCU 8051*

TMOD

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
← Timer 1 →				← Timer 0 →			

BIT	NOMBRE Y COMENTARIO
B0 - B1	MODO / M1 / M0 / MODO DE OPERACIÓN 0 / 0 / 0 / Temporizador de 13 *bits* 1 / 0 / 1 / Temporizador/Contador de 16 *bits* 2 / 1 / 0 / Tempoizador/Contador de 8 *bits* con auto-recarga 3 / 1 / 1 / Contadores múltiples específicos
B2	C/$\overline{T}$ Selecciona temporizador o contador • Si C/$\overline{T}$ = 0 entonces **temporiza** con los pulsos del reloj interno • Si C/$\overline{T}$ = 1 entonces **cuenta** los pulsos que llegan por T0 (pin14)
B3	GATE Habilita la entrada exterior $\overline{INT0}$ (pin 12) • Si GATE = 1 entonces habilita $\overline{INT0}$ si TR0 = 1 (control por *hardware*) • Si GATE = 0 entonces deshabilita $\overline{INT0}$ y depende exclusivamente de TR0 (control por *software*) (TR0 es un *bit* del Reg. TCON y se activa o desactiva por software)
B4 - B5 - B6 y B7	Configuración del *Timer* 1 Igual que para *Timer* 0, sustituyendo: T0 por T1 (pin 15) INT0 por INT1 (pin 13) TR0 por TR1

Figura 30.7 *Registro de control de modo (TMOD) de los timers 0 y 1 del 8051*

TCON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

BIT	NOMBRE Y COMENTARIO
B0	IT0 Control interrupción externa 0 (INT0) • Si IT0 = 0 entonces es activa por nivel bajo • Si IT0 = 1 entonces es activa por flanco de bajada
B1	IE0 Flag de interrupción para la interrupción externa 0 (INT0) • Se pone a uno cuando se detecta interrupción externa • Se repone automáticamente al atender la interrupción por flanco
B2	IT1 Control interrupción externa 1 (INT1)
B3	IE1 Falg de interrupción para la interrupción externa 1(INT1)
B4	TR0 Habilita temporizador/contador0 • Si TR0 = 1 entonces habilita temporizador/contador 0 • Si TR0 = 0 entonces deshabilita temporizador/contador 0
B5	TF0 Flag overflow (sobrepasamiento) del Timer0 Se repone automáticamente al atender la interrupción
B6	TR1 Habilita temporizador/contador 1
B7	TF1 Flag de overflow (sobrepasamiento) del Timer 1

Figura 30.8 *Registro de control (TCON) de los timers 0 y 1 del 8051*

1. Modo 0: temporizador-contador de 13*bits*

Al poner los *bits* M0 y M1 en 0 lógico en el registro TMOD (que no es direccionable *bit* a *bit*), se configura el **modo 0**; en éste el registro contador es de 13*bits*, repartidos los 8*bits* de mayor peso en THx y los 5*bits* más bajos en TLx. Cuando la cuenta en el registro del contador (THx,TLx) se desborda (pasa de $FFF1 a $0000) la bandera de interrupción correspondiente se activa, ésta puede ser TF0 ó TF1, y está ubicada en las posiciones de los *bits* 5 ó 7 del registro de control TCON, que hace parte de los SFR y que es direccionable *bit* a *bit*.

Como puede observarse en la figura 30.6, cualquier señal de interrupción externa (INTx) puede utilizarse para habilitar y "congelar" el valor de la cuenta del *timer*, al ponerse en 1 y 0 lógico respectivamente; y, siempre y cuando, los *bits* TRx y GATE permanezcan en 1.

2. Modo 1: temporizador-contador de 16*bits*

El **modo 1** se configura con los *bits* M0 y M1 en 1 y 0 respectivamente, y su comportamiento es idéntico al del modo 0, exceptuando que el registro contador del *timer* ahora es de 16*bits* y está ubicado en dos grupos de 8*bits* que residen en los registros THx y TLx. En este modo, el desbordamiento se da cuando pasa de $FFFF a $0000.

3. Modo 2: temporizador-contador de 8*bits* con recarga automática

El **modo 2** se configura con los *bits* M0 y M1 en 0 y 1 respectivamente; en este modo, los *timers* se comportan como contadores de 8*bits* y operan sobre la parte baja del registro contador (TLx).

Cuando el registro TLx se desborda (pasa de $FF a $00), se activa la bandera de interrupción adecuada (TF0 ó TF1) y el registro TLx se recarga o actualiza automáticamente con una copia del contenido de THx, previamente almacenado por el programador.

En la **figura 30.9** se muestra la arquitectura que adoptan los *timers* en modo 2.

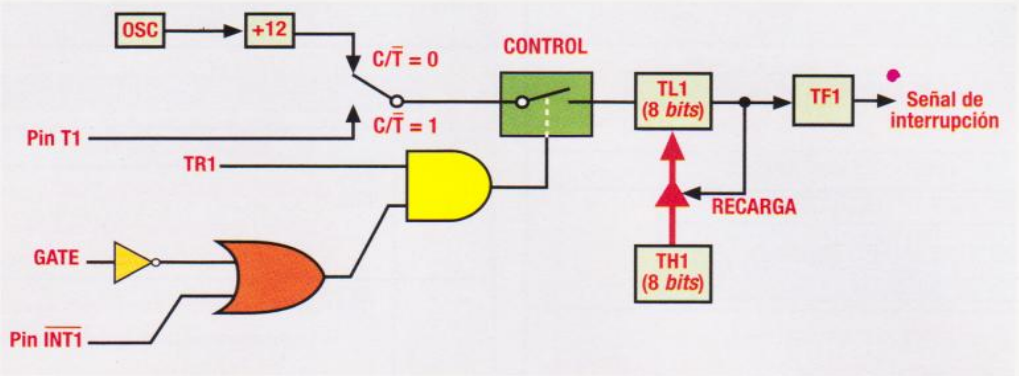


Figura 30.9 *Arquitectura equivalente de los timers del MCU 8051 en modo 2*

4. Modo 3: contadores múltiples

Con el **modo 3** se extiende a tres el número de *timers* disponibles en el microcontrolador 8051, éste se configura con los *bits* M0 y M1 en 1 y ha sido diseñado para las tareas que requieran otro temporizador-contador adicional; en este caso, se asocian los *timers* 0 y 1 para conformar dos contadores independientes de 8*bits* que utilizan como registros a las partes alta y baja del *timer* 0 (TH0 y TL0).

En la **figura 30.10** se muestra la arquitectura que asumen los nuevos *timers* independientes en modo 3, en ella puede verse que el *timer* 1 cede su bandera (TF1) y su *bit* de control (TR1) para conformar el contador adicional.

Con el registro TL0 se puede configurar un temporizador o contador interno o externo de 8*bits* con la lógica de control, tal como se ha considerado hasta ahora, pero, cuando se usa el registro TH0 (del contador adicional y también de 8*bits*), sólo se puede temporizar o contar tomando como base a 1/12 de la frecuencia del oscilador interno de la CPU.

El modo de operación de estos dos nuevos temporizadores-contadores es idéntico al de los anteriores, éstos activan sus correspondientes banderas de interrupción cuando sobre cualesquiera de los registros de los contadores (TL0 ó TH0) se produce un sobrepasamiento del valor $FF.

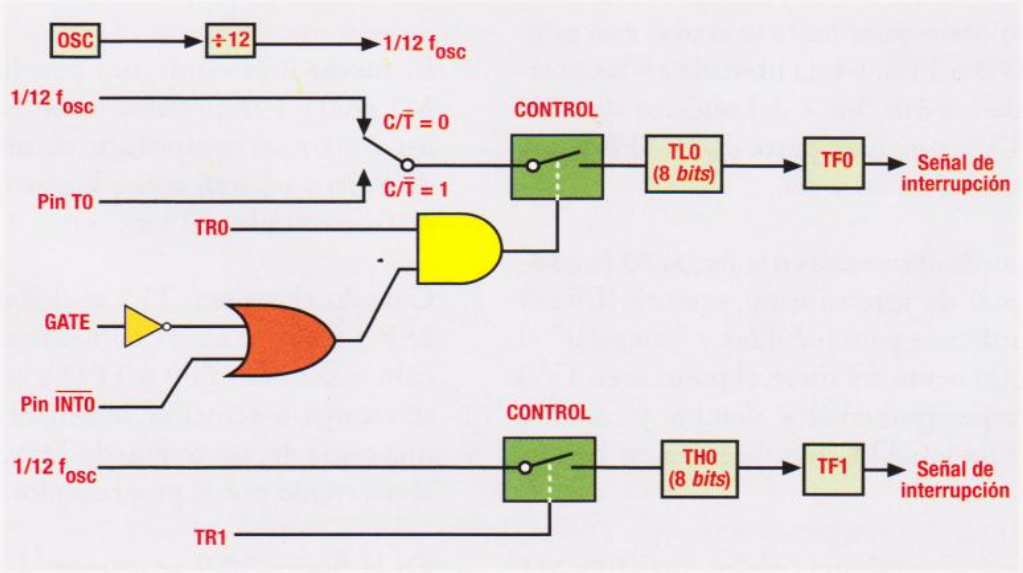
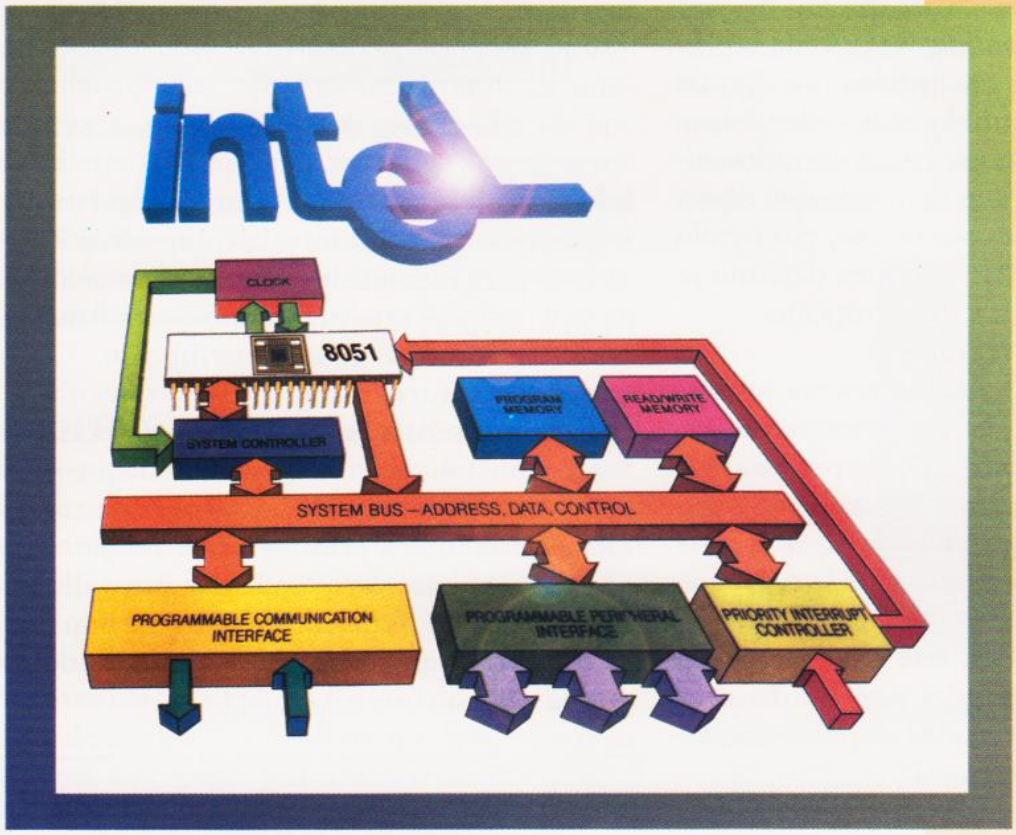


Figura 30.10 *Arquitectura equivalente de los timers del MCU 8051 en modo 3*



INTERRUPCIONES DEL MICROCONTROLADOR 8051

Las interrupciones en los procesos con microcontroladores son las herramientas que establecen un servicio directo entre la CPU y cualquier sistema o dispositivo periférico externo e interno, lo que nos permite desarrollar un modo de trabajo que es inherente al control de procesos en tiempo real.

Los microcontroladores 8051 son diseñados con módulos periféricos internos especiales que pueden ser programados, y que reciben y evalúan todas las fuentes de interrupción internas y externas.

Introducción

Tradicionalmente el servicio bidireccional no sincrónico y sincrónico entre la CPU del microcontrolador y los dispositivos y sistemas periféricos, se ha desarrollado por medio de técnicas de consulta conocidas como ***polling*** y el uso de protocolos de comunicación construidos con algunas líneas de diálogo, también conocidos como ***handshake***. En este caso se hacen consultas cíclicas de los registros o *bits* de estado que representan a cada uno de los periféricos, por medio de instrucciones del programa, para determinar cuando se ha solicitado una interrupción.

No obstante, este método presenta problemas cuando los periféricos consultados son numerosos, porque, en cada ciclo de programa se deben interrogar y actualizar los registros o *bits* de consulta, y, porque al dispositivo sólo se le atiende al terminar la ejecución de la rutina de consulta y no en el momento de la solicitud de atención. Por lo anterior, este método sólo se usa con eficiencia en grandes sistemas de cómputo que poseen alta velocidad de procesamiento y alta capacidad de memoria.

Sin embargo, las **interrupciones** son otra técnica que viene siendo utilizada por todos los fabricantes de microcontroladores como una herramienta poderosa para establecer un servicio no sincrónico, directo e inmediato, entre la CPU y los sistemas o dispositivos periféricos externos e internos. Con esta técnica se evita la ejecución de las rutinas de consulta y se atienden los procesos en tiempo real, ya que el periférico genera una señal de interrupción (INTx) propia, cada vez que requiere atención. Este método nos permite ejecutar las rutinas de servicio de las interrupciones en el momento preciso y sin desviar la atención de la CPU del cuerpo principal del programa, lo cual se hace necesario en muchas de las tareas de los microcontroladores.

Las interrupciones del MCU 8051

Los microcontroladores 8051 cuentan con cinco fuentes de interrupción, dos externas y tres internas. En la **tabla 31.1** se muestran las interrupciones, sus nombres y los mnemónicos que se usan para cada una de ellas; y, en la columna de la derecha se muestran también las banderas que están asociadas a cada interrupción.

NOMBRE	MNEMÓNICO	BANDERA
Externa 0	$\overline{INT0}$	IE0
Timer 0 (Interna)	TIMER0	TF0
Externa 1	$\overline{INT1}$	IE1
Timer 1 (Interna)	TIMER1	TF1
Puerto serial (Interna)	RI	RI
Puerto serial (Interna)	TI	TI

Tabla 31.1 *Las interrupciones del MCU 8051*

Todas las interrupciones pueden ser habilitadas e inhabilitadas independientemente y globalmente por el usuario, usando el registro habilitador de interrupciones **IE** (*interrupt enable register*) que hace parte de los registros de función especial SFR; en el **cuadro 31.1** se muestra este registro con todas las condiciones de sus *bits* de control.

IE (REGISTRO HABILITADOR DE INTERRUPCIONES)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EA	X	X	ES	ET1	EX1	ET0	EX0

BIT	NOMBRE Y COMENTARIO
B0	**EX0** • Si EX0 = 1 habilita interrupción externa INT0 • Si EX0 = 0 la inhabilita
B1	**ET0** • Si ET0 = 1 habilita interrupción del *Timer* 0 • Si ET0 = 0 la inhabilita
B2	**EX1** • Si EX1 = 1 habilita interrupción externa INT1 • Si EX1 = 0 la inhabilita
B3	**ET1** • Si ET1 = 1 habilita interrupción del *Timer* 1 • Si ET1 = 0 la inhabilita
B4	**ES** • Si ES = 1 habilita interrupción del puerto serial • Si ES = 0 la inhabilita
B5	Reservado
B6	Reservado
B7	**EA** • Si EA = 1 habilita individualmente a todas las interrupciones que en este registro están a *uno* • Si EA = 0 no reconoce ninguna interrupción

Cuadro 31.1 *Registro habilitador de las interrupciones IE del 8051*

Cada una de las interrupciones afecta a un determinado *bit* bandera (*flag bit*) cada vez que es solicitada, y una vez se han atendido las rutinas de interrupción el programador deberá estar pendiente del estado de éstas y retornarlas a su estado lógico inicial para poder aceptar la próxima interrupción. Todas las banderas de las interrupciones pueden borrarse por medio de instrucciones (borrado por *software*) directamente en los registros que las contienen TCON y SCON, aunque algunas se borran de manera automática al retornar de la interrupción (borrado por *hardware*).

En el caso de las dos interrupciones externas **INT0** e **INT1**, que se pueden activar por flanco o por nivel a través de los *bits* de control de interrupción **IT0** e **IT1** del registro TCON, las banderas son borradas automáticamente cuando están activas por flanco; pero, cuando están activas por nivel bajo, se deben inhabilitar (poner en uno lógico el correspondiente pin de entrada INTx) para salir de la rutina de servicio de atención a la interrupción y borrar la bandera. Las interrupciones internas de los temporizadores-contadores (*timers*) y las del puerto serial, solamente pueden ser borradas por el usuario a través del *software*. En la **figura 31.1** se aprecian las fuentes de interrupción del 8051 y sus selectores de entrada.

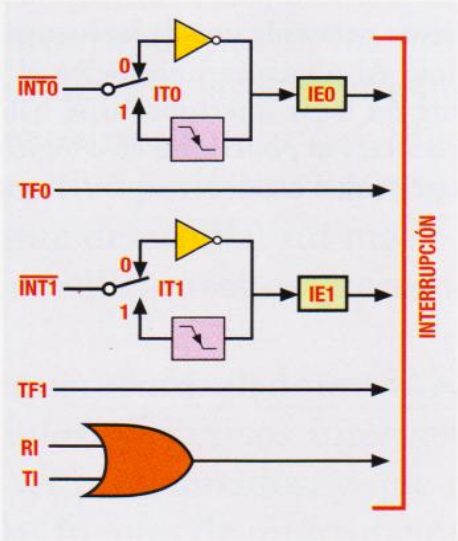


Figura 31.1 *Fuentes de interrupción del 8051, selectores y banderas*

Vectorización de las interrupciones

Cada vez que se acepta una interrupción, la CPU guarda el contenido del registro contador del programa PC en el área de la pila (*stack*) antes de que éste sea cargado con el vector correspondiente (dirección fija en memoria de programa asociada con la interrupción); de esta forma, una vez atendida la interrupción, la CPU puede continuar con la ejecución del programa principal desde el punto en el cual fue interrumpido.

Figura 31.2 *Vectores de las interrupciones del 8051*

En la **figura 31.2** se muestra la vectorización de todas las interrupciones del 8051, esta área se ubica al comienzo de la memoria de programa y allí puede verse que se han reservado espacios de 8 *bytes* para cada una de ellas; estos intervalos de 8 *bytes* entre cada vector de interrupción pueden utilizarse para contener directamente los programas cortos de atención a las interrupciones, o, como es habitual cuando la rutina de servicio de interrupción es extensa, para albergar la instrucción de salto **JMP**(dir) necesaria para cargar el registro PC con la dirección de memoria inicial de la rutina.

En este último caso y a manera de ejemplo, si suponemos que la rutina de atención a la interrupción se encuentra ubicada a partir de la posición $0777 de la memoria de programa, y que además, la interrupción externa INT1 se ha habilitado poniendo el *bit* EX1 en uno lógico, el vector correspondiente, que está ubicado en la posición $0013, debe contener el código de ope-

ración de la instrucción de salto y, en los dos *bytes* de memoria siguientes $0014 y $0015, deben guardarse los *bytes* de la dirección de salto más y menos significativos respectivamente. En la **figura 31.3** se representa este ejemplo.

Figura 31.3 *Salto a una rutina de interrupción en memoria de programa*

Para comprender el proceso de interrupción en los microcontroladores 8051, se ha incluido el diagrama de tiempos simplificado bajo condiciones ideales que se muestra en la **figura 31.4**, éste está representado en términos de los estados (S) y de las fases (P) de cada ciclo de máquina (C). En el diagrama puede verse que las banderas de interrupción se consultan en la fase 2 del estado 5 (S5P2) de cada ciclo de máquina y que los resultados son evaluados durante el siguiente ciclo; y también, que luego de la evaluación y si hay alguna solicitud de interrupción activa se genera un llamado (LCALL) al vector correspondiente.

El proceso de interrupción siempre se considera urgente y, una vez iniciada su vectorización, sólo podrá ser interrumpida o bloqueada cuando una instrucción o una interrupción de mayor o igual prioridad se esté ejecutando, o cuando se esté accediendo a los registros de control IP ó IE, ó, cuando se esté ejecutando una instrucción de retorno de interrupción RETI.

Nota:
Toda rutina de atención a una interrupción debe terminar con la instrucción de retorno de interrupción **RETI**; con ésta se informa a la CPU que la rutina ha concluido para que descargue de la pila la dirección de dos *bytes* a partir de la cual debe continuar la ejecución del programa principal, y la cargue en el registro contador de programa (PC). Las rutinas de servicio de las interrupciónes deben hacerse breves, para garantizar que sólo se experimentará una pérdida de tiempo relativamente pequeña durante la ejecución del programa principal al momento de atender la interrupción.

Orden de prioridad de las interrupciones

En el **cuadro 31.2** se muestra el registro para el control de prioridad de las interrupciones **IP** (*interrupt priority register*) que también hace parte de los SFR; con él se pueden programar dos niveles de prioridad (alta y baja) de manera independiente para cada una de las fuentes de interrupción. Al poner en uno lógico el correspondiente *bit* de la interrupción se habilita la prioridad alta, y al ponerlo en cero lógico se habilita la prioridad baja.

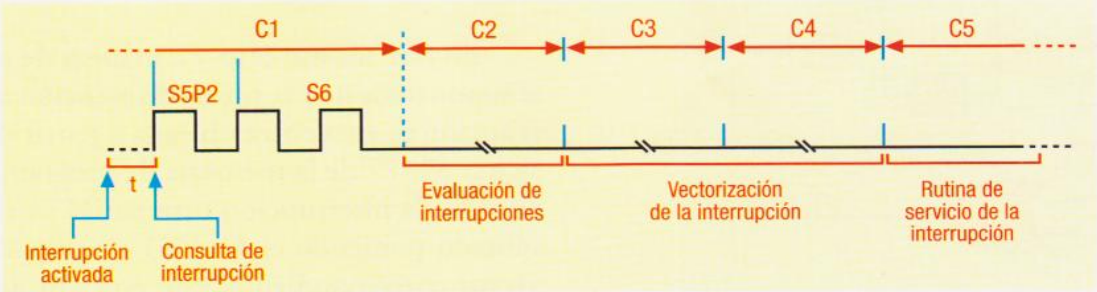


Figura 31.4 *Diagrama de tiempos de un ciclo ideal del proceso de interrupción*

IP (REGISTRO DE PRIORIDAD DE LAS INTERRUPCIONES)							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
X	X	X	PS	PT1	PX1	PT0	PX0

BIT	NOMBRE Y COMENTARIO
B0	PX0 • Si PX0 = 1 define alta prioridad a la interrupción INT0
B1	PT0 • Si PT0 = 1 define alta prioridad a la interrupción del *Timer* 0
B2	PX1 • Si PX1 = 1 define alta prioridad a la interrupción INT1
B3	PT1 • Si PT1 = 1 define alta prioridad a la interrupción del *Timer* 1
B4	PS • Si PS = 1 define alta prioridad a la interrupción del puerto serial
B5	Reservado
B6	Reservado
B7	Reservado

Cuadro 31.2 *Registro para el control de prioridad de las interrupciones IP del 8051*

En cualquier caso en el cual sean solicitadas más de dos interrupciones en el mismo instante, la CPU atenderá las solicitudes de interrupción y evaluará su nivel de prioridad para determinar cuál deberá ser atendida primero, ésta será la que tenga mayor prioridad o la que corresponda, de acuerdo con la **tabla 31.2**, que ha sido predeterminada para ser consultada por el microcontrolador 8051 en todos aquellos casos en los cuales las interrupciones presenten el mismo nivel de prioridad.

Cuando una interrupción tiene nivel de prioridad bajo puede ser interrumpida, a su vez, únicamente por otra con nivel de prioridad alto; mientras que una interrupción con nivel de prioridad alto sólo podrá ser interrumpida por el RESET, que goza del nivel máximo de prioridad.

NIVELES DE PRIORIDAD PREDETERMINADOS

PRIORIDAD	BANDERA	INTERRUPCIÓN
(más alta)... 1	$\overline{INT0}$	EXTERNA 0
2	TIMER0	TIMER0
3	$\overline{INT1}$	EXTERNA 1
4	TIMER1	TIMER1
(más baja)... 5	RI y TI	PUERTO SERIAL

Tabla 31.2 *Niveles de prioridad predeterminados para el 8051*

PARA SABER MÁS:
Simulación por software de un tercer nivel de prioridad

En aquellas tareas que utilizan varias fuentes de interrupción con diferentes niveles de jerarquía, es conveniente contar con más de dos niveles de prioridad, en estos casos, se puede crear virtualmente un tercer nivel y simular su comportamiento. El proceso necesario para crear el tercer nivel de prioridad se describe a continuación.

Primero, se deben elegir las interrupciones que van a tener el nivel más alto de prioridad, superior a 1 (prioridad alta), y se les debe asignar la prioridad 1 a través del registro IP. Las rutinas de servicio de las interrupciones con prioridad 1, que se supone que deben ser interrumpibles por interrupciones con nivel de prioridad 2, se acompañan por los siguientes segmentos de código de programa al inicio y antes de la instrucción RETI:

```
        PUSH    IE
        MOV     IE,#MASCARA
        CALL    RETORNO
*****************************************************
 RUTINA DE SERVICIO DE LA INTERRUPCIÓN
*****************************************************
        POP     IE
        RET
RETORNO: RETI
```

Este programa adicional se utiliza para que, una vez sea reconocida cualquier solicitud de interrupción prioritaria, el registro habilitador de interrupciones IE se defina nuevamente con el valor de la MASCARA, de forma tal, que inhabilite todas las interrupciones menos las de prioridad 2; y luego, por medio del llamado a la etiqueta RETORNO finalice ejecutando la instrucción RETI (lo cual borrará la interrupción con prioridad 1 en progreso). En este punto cualquier interrupción con prioridad 1 que esté habilitada podrá ser atendida.

Con la instrucción de descarga de la pila (POP) se recupera el contenido original del *byte* del registro habilitador IE, y finalmente, con una instrucción de retorno de subrutina (RET) se termina la rutina de servicios de la interrupción. Este *software* adicional que se requiere no retarda considerablemente la ejecución de la rutina de servicio de la interrupción con prioridad 1, puede calcularse un retardo de apenas 10ms cuando el oscilador del microcontrolador tiene una frecuencia de 12MHz.

Interrupciones internas de los TIMER 0 y 1

Puesto que los *timers* y sus interrupciones ya se estudiaron en la lección anterior, aquí sólo recordaremos brevemente que las banderas TF0 y TF1 de éstos, son las encargadas de generar la solicitud de interrupción respectiva a la CPU y que se ponen en 1 lógico, cuando se desbordan los correspondientes registros de conteo.

En la **figura 31.5** se muestra un resumen de los registros contadores de los *timers* en los cuatro modos de operación, y las banderas de interrupción que se afectan en cada caso.

Como un resumen de los principales aspectos que se han estudiado de las interrupciones tanto internas como externas, a continuación, en la **figura 31.6**, se muestra un esquema simplificado de la arquitectura del sistema de interrupciones del microcontrolador 8051; allí se representan el *hardware* y la lógica asociada con cada una de las fuentes de interrupción, y con lógica simple, la función de los registros de control de las interrupciones (IP e IE).

Figura 31.5 *Banderas de interrupción de los timers del 8051*

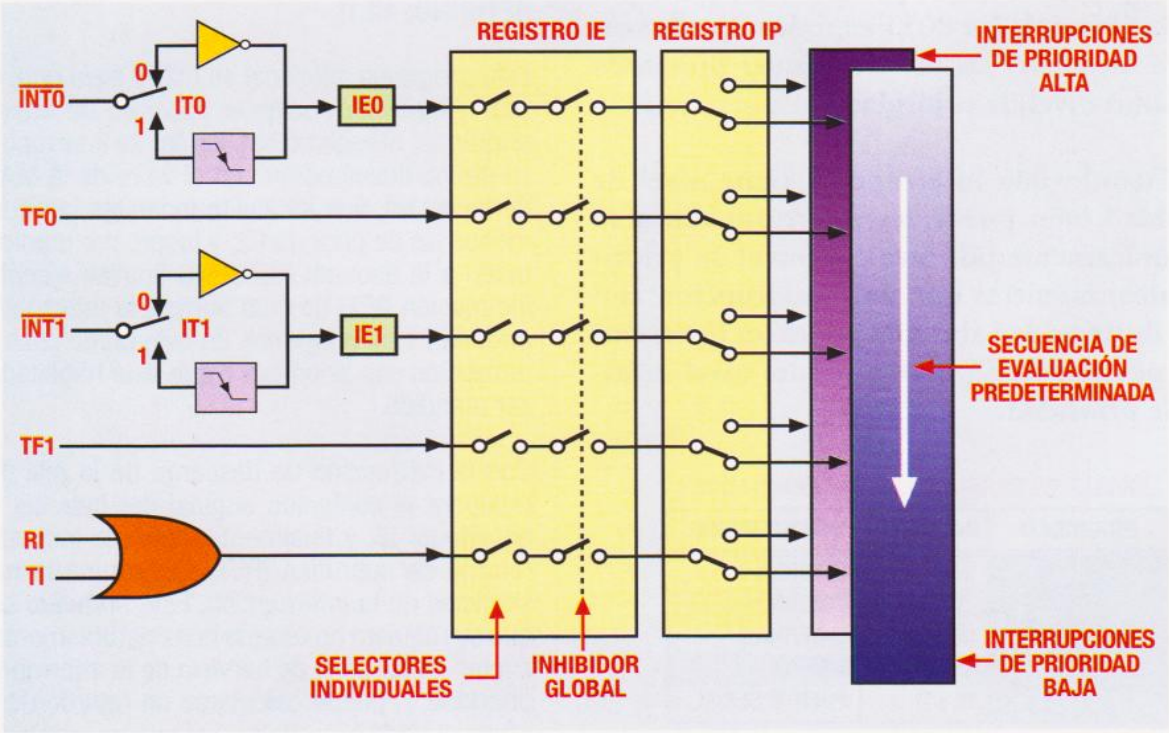
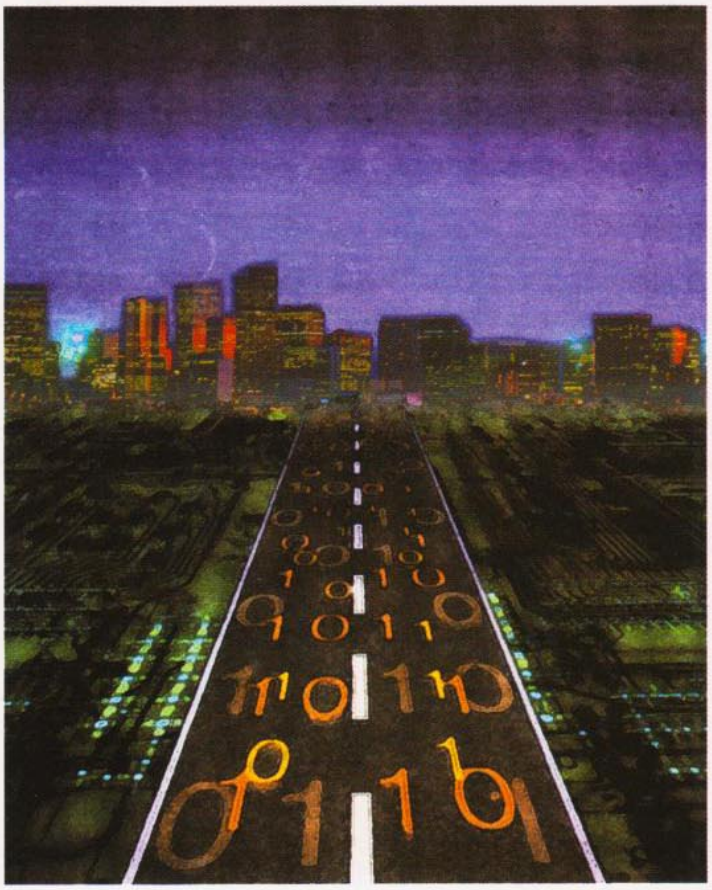


Figura 31.6 *Arquitectura simplificada del sistema de interrupciones*



COMUNICACIONES SERIALES CON EL MICROCONTROLADOR 8051

Los módulos de comunicación internos en los microcontroladores mejoran su flexibilidad porque facilitan la compatibilidad y el intercambio de datos entre ellos y otros dispositivos o sistemas electrónicos.

Los microcontroladores 8051 poseen un puerto de comunicación serial programable y flexible, que nos permite configurar la transmisión y recepción simultánea y no sincrónica de datos, en cuatro modos.

El UART del microcontrolador 8051

Los microcontroladores 8051 cuentan con un puerto de comunicaciones serial no sincrónico y *full-duplex* (que puede recibir y transmitir simultáneamente datos), que generalmente es conocido como un receptor y transmisor universal no sincrónico, o UART (*universal asynchronous receiver and transmiter*). Éste es programable en uno de cuatro modos de comunicación posibles (MODO0 a 3) por medio de los *bits* 7 y 6, **SM0** y **SM1**, del registro de control y de estado del puerto serial **SCON** (*serial port control register*) que hace parte de los SFR, tal como se muestra en el **cuadro 32.1**.

Para recibir y transmitir los datos el UART cuenta con el registro especial **SBUF** que se denomina el **buffer de datos serial**, y que se localiza en la posición de memoria $99 de los SFR. Al escribir en este registro, se carga el dato que se va a transmitir (un *byte*), y leyéndolo, se accede al dato de un *byte* recibido (en la **figura 32.1** se ha representado el SBUF). En cualquiera de los modos la transmisión se inicia cuando se usa el registro SBUF como destino por medio de cualquier instrucción, mientras que la recepción se inicia en MODO0 cuando se ponen los *bits* **RI** y **REN** en 0 y 1 respectivamente; y en los MODOS1,2 y 3, cuando **REN** está en 1 y se recibe el ***bit* de inicio** (***start bit***).

SCON (REGISTRO DE ESTADO Y CONTROL DEL PUERTO SERIAL

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

BIT	NOMBRE Y COMENTARIO
B0	**RI** Bandera de interrupción de la recepción Se activa por *hardware* al finalizar la recepción del 8º *bit* en el Modo 0 o hacia la mitad del intervalo de tiempo del *bit* de *Stop* en los otros Modos (excepto ver SM2) Debe ser desactivado por *software*
B1	**TI** Bandera de interrupción de la transmisión Se activa por *hardware* al final de la transmisión del 8º *bit* en el Modo 0 o al comienzo del *bit* de *Stop* en los otros Modos Debe ser desactivado por *software*
B2	**RB8** En los Modos 2 y 3 es el 9º *bit* que se recibe En Modo 1, si SM2 = 0, RB8 es el bit de *Stop* En Modo 0 no se utiliza
B3	**TB8** Corresponde al 9º *bit* de datos en los Modos 2 y 3 Es programable por el usuario . Habitualmente es el *bit* de paridad
B4	**REN** • Si REN = 1 (por *software*) habilita la recepción • Si REN = 0 no recibe
B5	**SM2** • En Modo 2 y 3, si SM2 = 1 entonces RI no se activará si el 9º *bit* de datos (RB8) es igual a cero • En Modo 1, si SM2 = 1 entonces RI no se activará si el *bit* de *Stop* no se ha recibido • En Modo 0, SM2 debe estar en cero
B6 - B7	**SM0 - SM1** Especifica el Modo según la tabla

SM0	SM1	MODO	DESCRIPCIÓN	VELOCIDAD
0	0	0	Desplaza 8 *bits*	Oscilador/12
0	1	1	UART de 8 *bits*	Programable
1	0	2	UART de 9 *bits*	Oscilador/64 o Oscilador/32
1	1	3	UART de 9 *bits*	Programable

Cuadro 32.1 *Registro de control y de estado del puerto serial SCON del 8051*

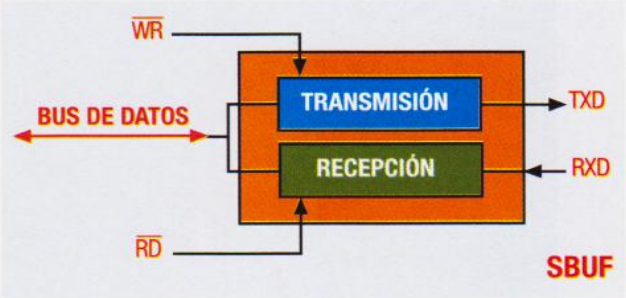


Figura 32.1 *Buffer de datos serial del puerto de comunicaciones del 8051*

Configuración de la velocidad de comunicación

La velocidad de la comunicación, en *bits* por segundo o baudios, está en función del modo de comunicación activo, así: en MODO0, la velocidad se determina simplemente por la **ecuación 32.1** que divide la frecuencia del oscilador de la CPU entre 12; en MODO2 está en función del valor del *bit* 7, **SMOD**, del registro de control de potencia **PCON** (*power control register*) que pertenece a los SFR. Cuando el *bit* duplicador de baudios (SMOD) está en cero, la velocidad es 1/64 de la frecuencia del oscilador, y cuando está en uno, es 1/32 de la frecuencia del oscilador; ésto se expresa en la **ecuación 32.2**. En el **cuadro 32.2** se muestra el registro PCON.

$$\text{BAUDIOS en Modo 0} = \frac{\text{Frecuencia del oscilador}}{12}$$

Ecuación 32.1 *Velocidad de comunicación en MODO0*

$$\text{BAUDIOS en Modo 2} = \frac{2^{SMOD}}{64}\ (\text{Frecuencia del oscilador})$$

Ecuación 32.2 *Velocidad de comunicación en MODO2*

$$\text{BAUDIOS en Modos 1 y 3} = \frac{2^{SMOD}}{32}\ \frac{\text{Frecuencia del oscilador}}{12 \times (256\text{-TH1})}$$

Ecuación 32.3 *Velocidad de comunicación en los MODOS1 y 3 con el TIMER1*

PCON (REGISTRO DE CONTROL DE POTENCIA)							
Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
SMOD	-	-	-	GF1	GF0	PD	IDL

BIT	NOMBRE Y COMENTARIO
B0	**IDL** *Bit* Modo Idle (ralentizado) Si IDL = 1 se activa
B1	**PD** *Bit Power Down* (baja potencia) Si PD = 1 se activa
B2	**GF0** *Bit* bandera de propósito general
B3	**GF1** *Bit* bandera de propósito general
B4 - B5-B6	Reservados
B7	**SMOD** *Bit* duplicador de baudios Si SMOD = 1 se duplica la frecuencia de reloj del *timer* 1 cuando éste se utiliza como generador de baudios en los modos de comunicación 1, 2 y 3

- Si se activan simultáneamente PD e IDL el microcntrolador asume el Modo *Power Down*, que tiene prioridad sobre el Modo Idle
- Después de la operación *Reset*, PCON toma el siguiente valor 0XXX0000
- En los dispositivos HMOS, PCON contiene solamente el *bit* SMOD

Cuadro 32.2 *Registro de control de potencia PCON del 8051*

En MODO1 y MODO3 la velocidad de la comunicación está en función del estado del *bit* SMOD y del valor de recarga y sobrepasamiento del registro contador del TIMER1, el cual debe estar configurado como temporizador de 8 *bits* con recarga automática (en modo 2) para optimizar su uso, y extenderlo a la mayoría de las tareas; tal como se muestra en la **ecuación 32.3**

En la **tabla 32.1** se muestra una lista con las velocidades de comunicación serial estandarizadas y la manera de obtenerlas en los MODOS1 y 3, en ésta se relaciona la velocidad con la frecuencia del oscilador del microcontrolador, con el estado del *bit* SMOD y con la configuración del TIMER1.

MODO0 de comunicación

Con los *bits* SM0 y SM1 puestos en cero, se activa el MODO0 de comunicación serial, en éste la entrada y salida de los datos se hace a través del pin 10 (**RXD**) del microcontrolador, mientras que por el pin11 (**TXD**) se generan los pulsos de desplazamiento (*shift clock*); la transmisión o recepción del dato de 8 *bits* se hace empezando por el *bit* menos significativo (LSB) y a una velocidad fija e igual a 1/12 de la frecuencia del oscilador. En la **figura 32.2** se muestra la arquitectura simplificada que adopta el puerto serial cuando se activa el MODO0, y en la **figura 32.3** se muestra el diagrama de tiempos (o cronograma) que le corresponde y que además describe detalladamente el proceso de transmisión y recepción en dicho modo.

En el diagrama de tiempos se puede apreciar que la transmisión se inicia cuando se usa el registro SBUF como destino, y que además la señal de control interna **SEND** habilita la salida en serie del dato del *buffer* y la salida de la señal de desplazamiento (SHIFT). Cada vez que se envía un *bit* del dato del *buffer*, ingresa un cero, y de esta forma, al transmitir el *bit* más significativo (MSB) el *buffer* serial queda cargado con ceros.

VELOCIDAD (BAUDIOS)	FRECUENCIA DEL OSCILADOR	SMOD	*TIMER1*		
			C/T	MODO	VALOR DE RECARGA (TH1)
Modo 0, Max: 1 MHz	12 MHz	X	X	X	X
Modo 2, Max: 375 K	12 MHz	1	X	X	X
Modos 1 y 3:62,5 K	12 MHz	1	0	2	$FF
19,2 K	11,059 MHz	1	0	2	$FD
9,6 K	11,059 MHz	0	0	2	$FD
4,8 K	11,059 MHz	0	0	2	$FA
2,4 K	11,059 MHz	0	0	2	$F4
1,2 K	11,059 MHz	0	0	2	$F8
137,5 K	11,986 MHz	0	0	2	$1D
110 K	6 MHz	0	0	2	$72
110 K	12 MHz	0	0	1	$FEEB

Tabla 32.1 *Velocidades de comunicación estándar en los MODOS1 y 3 con el TIMER1*

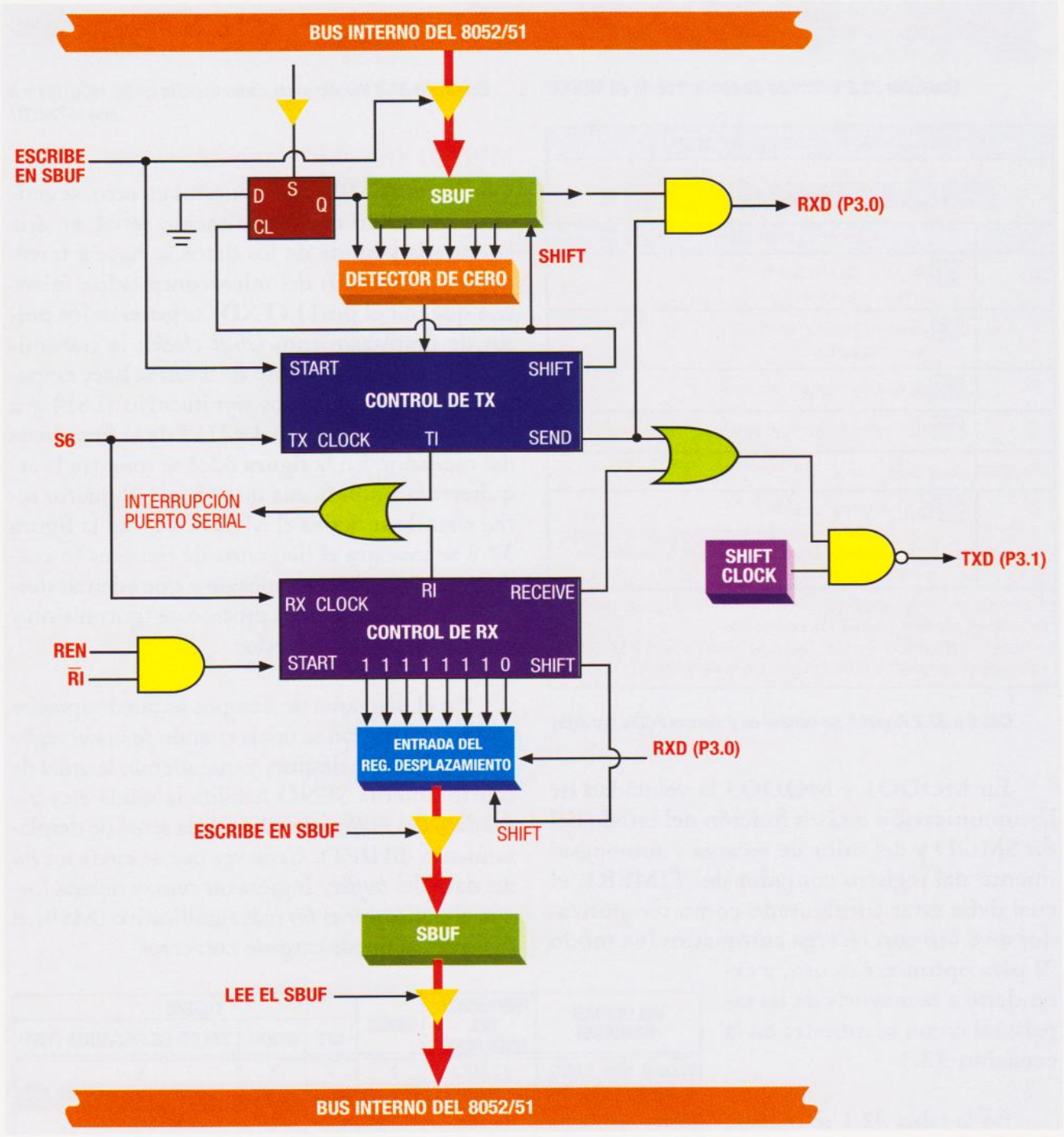


Figura 32.2 *Arquitectura del puerto serial en MODO0*

El proceso de recepción se inicia cuando REN vale 1 y RI vale 0, bajo esta condición se cargan unos en el *buffer* receptor y, un tiempo después, se activa la señal de control interna **RECEIVE** que habilita la señal de desplazamiento (SHIFT) y carga en serie los 8 *bits* del dato que van siendo muestreados por el pin 10 (RXD). En el décimo ciclo de máquina, tras haber puesto en 0 el *bit* RI, automáticamente la señal RECEIVE se borra y la bandera de interrupción de recepción RI es puesta en 1. En la **figura 32.4** se muestra el esquema simplificado de la comunicación y el formato de palabra en MODO0.

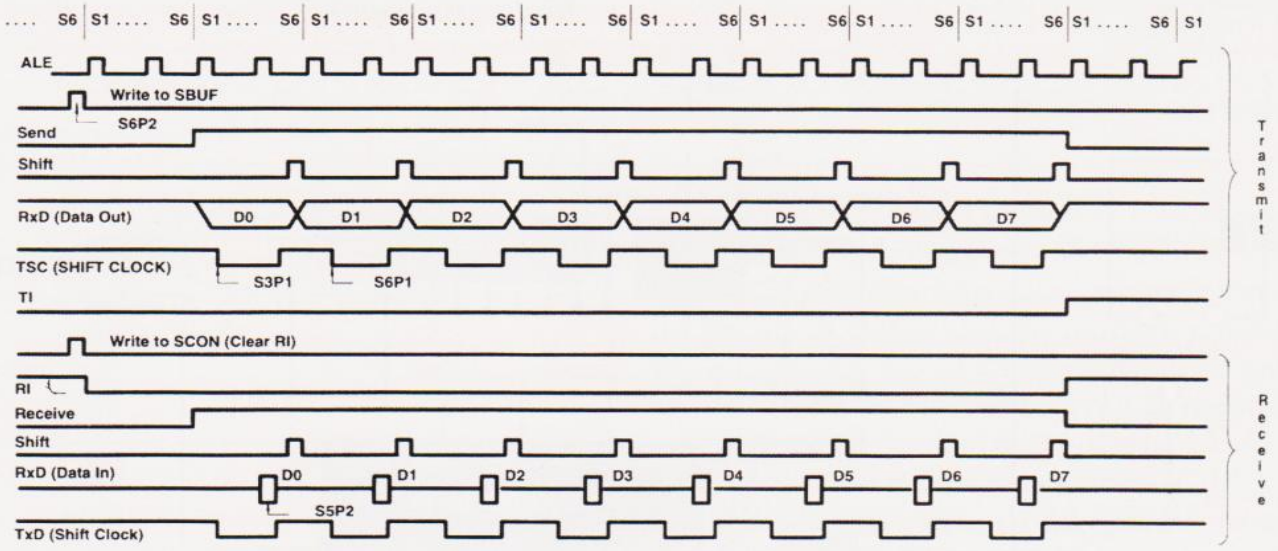


Figura 32.3 *Cronograma del proceso de comunicación en MODO0*

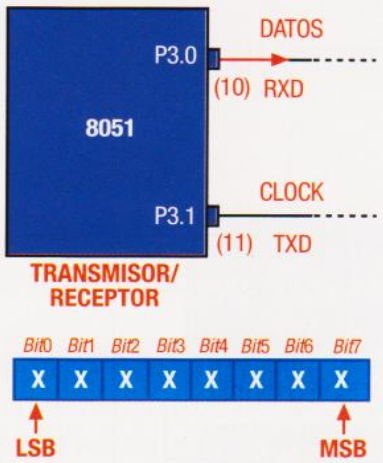


Figura 32.4 *Esquema de comunicación y formato de palabra en MODO0*

MODO1 de comunicación

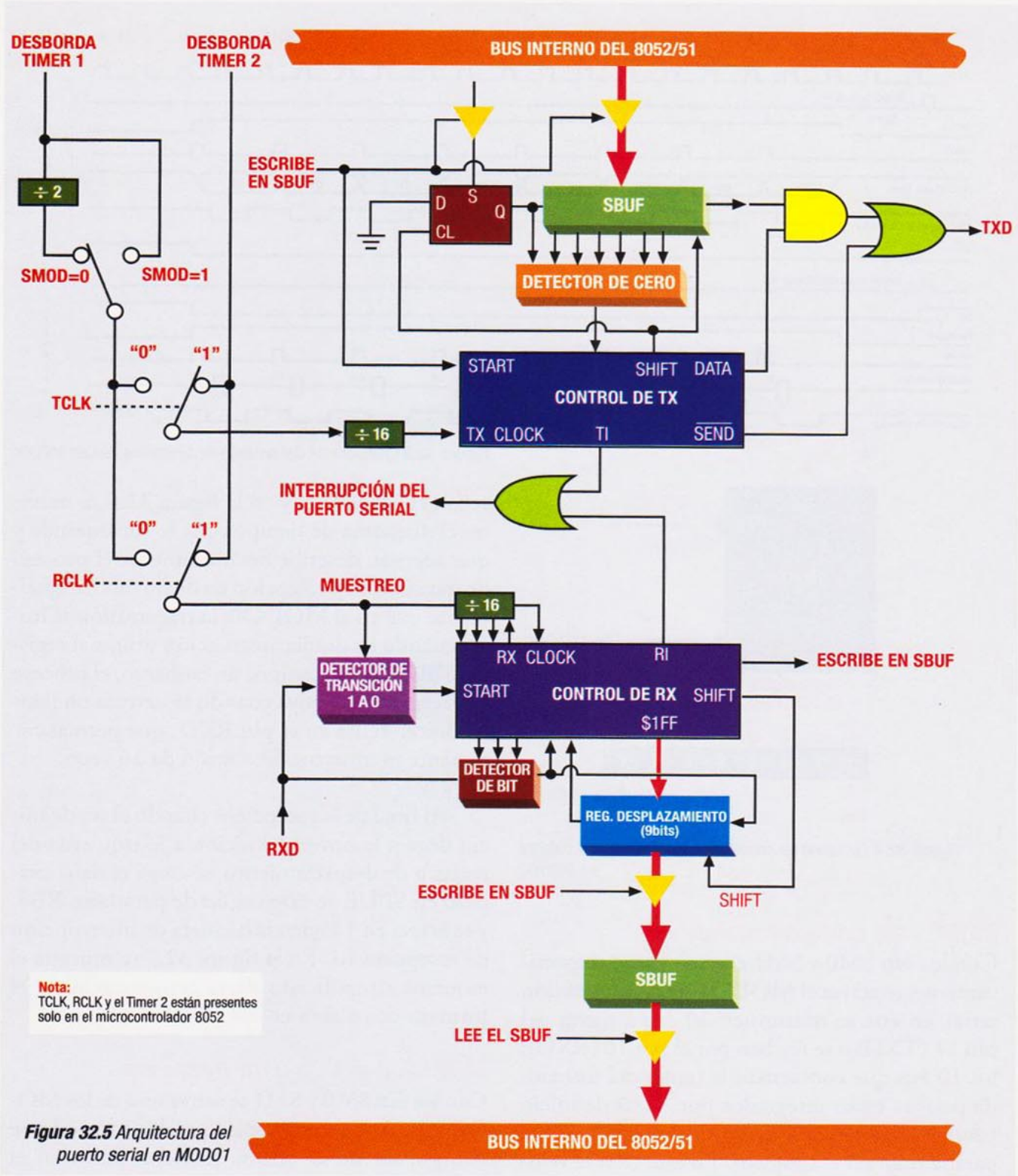
Con los *bits* SM0 y SM1 en cero y uno respectivamente, se activa el MODO1 de comunicación serial, en éste se transmiten 10 *bits* a través del pin 11 (TXD) o se reciben por el pin 10 (RXD); los 10 *bits* que conforman la trama del formato de palabra están integrados por: 1 *bit* de inicio (*start bit* = 0 lógico), 8 *bits* de datos y 1 *bit* de parada (*stop bit* = 1 lógico). En este caso la velocidad de la comunicación, como ya se estudió, está expresada en función del TIMER1.

La **figura 32.5** muestra la arquitectura simplificada que adopta el puerto serial cuando se activa el MODO1, y en la **figura 32.6** se muestra el diagrama de tiempos que le corresponde y que además, describe detalladamente el proceso de transmisión y recepción en dicho modo. Igualmente que en el MODO0, la transmisión se inicia cuando cualquier instrucción utiliza el registro SBUF como destino, sin embargo, el proceso de recepción se activa cuando se detecta un flanco descendente en el pin RXD, que permanentemente es muestreado a razón de 16 veces.

Al final de la recepción, cuando el *bit* de inicio llega a la novena posición a la izquierda del registro de desplazamiento, se carga el dato recibido en SBUF, se carga el *bit* de parada en RB8, y se activa en 1 lógico la bandera de interrupción de recepción RI. En la **figura 32.7** se muestra el esquema simplificado de la comunicación y el formato de palabra en MODO1.

MODOS 2 y 3 de comunicación

Con los *bits* SM0 y SM1 se activa uno de los MODOS 2 ó 3 de comunicación serial, estos modos se comportan de la misma manera, pero, en el MODO3 el programador puede ajustar la velocidad dentro de una amplia gama, como ya se estudió. En cualquiera de estos modos se transmiten 11 *bits* a través del pin 11 (TXD) o se reciben por el pin 10 (RXD), la trama de los 11 *bits* se conforma



Nota:
TCLK, RCLK y el Timer 2 están presentes solo en el microcontrolador 8052

Figura 32.5 *Arquitectura del puerto serial en MODO1*

por 1 *bit* de inicio (*start bit* = 0 lógico), 8 *bits* de datos, 1 noveno *bit* programable por el usuario como *bit* de paridad y 1 *bit* de parada (*stop bit* = 1 lógico).

La velocidad de la comunicación dependerá del modo de comunicación activo, en MODO2 la velocidad es fija e igual a 1/64 ó 1/32 de la frecuencia, mientras que en el MODO3 es variable, configurable y está en función del TIMER1, tal como ya se ha expresado en la ecuación 32.3.

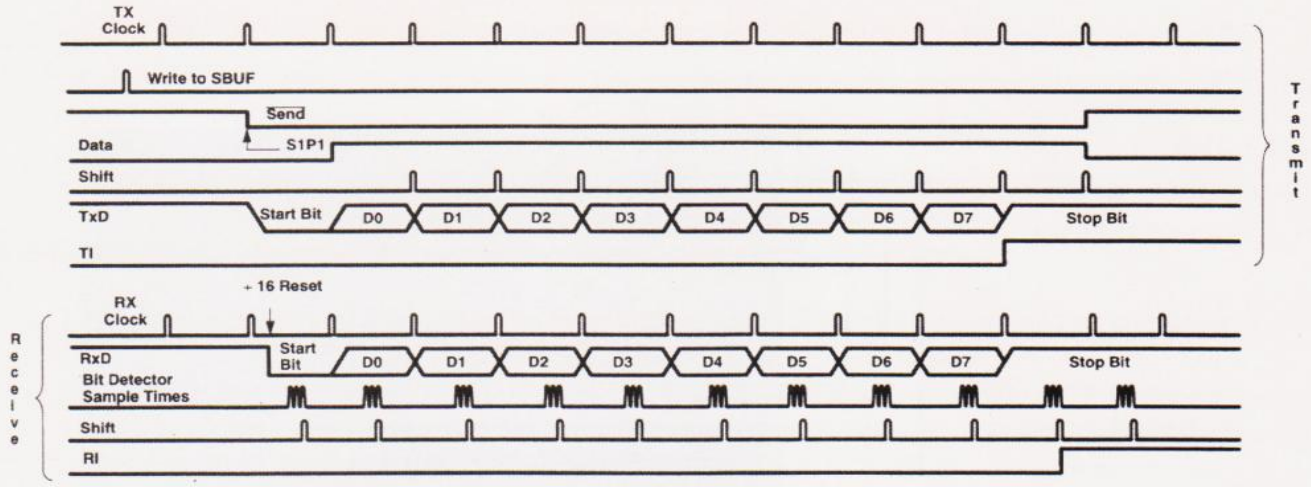


Figura 32.6 Cronograma del proceso de comunicación en MODO1

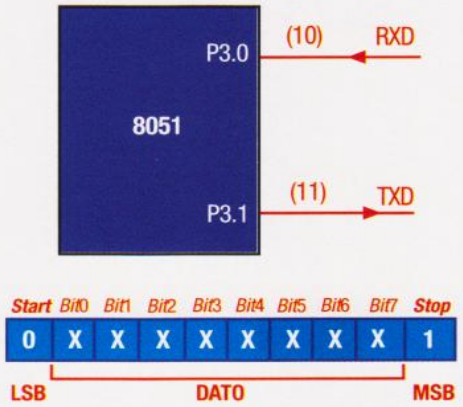


Figura 32.7 Esquema de comunicación y formato de palabra en MODO1

En la **figura 32.8** se muestra el cronograma que le corresponde y que además describe detalladamente el proceso de transmisión y recepción en el MODO2 de comunicación, mientras que en la **figura 32.9** se aprecia el esquema simplificado de la comunicación y el formato de palabra en los MODOS 2 y 3.

La **figura 32.10** muestra la arquitectura simplificada que adopta el puerto serial cuando se activa el MODO2 (los MODOS 1 y 3 poseen la misma arquitectura, ver figura 32.4)

De la misma forma que en los otros modos, la transmisión se inicia cuando cualquier instrucción utiliza al registro SBUF como destino, tras esta acción se carga el *bit* 3 (TB8) del registro SCON en la posición del noveno *bit* del registro de desplazamiento (como un *bit* de paridad) y luego, se da paso a la transmisión en serie.

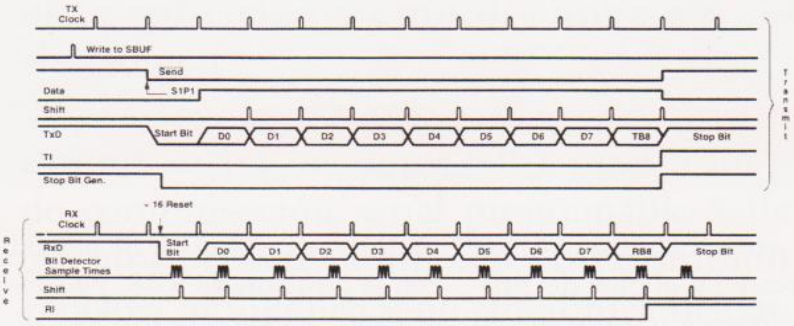


Figura 32.8 Cronograma del proceso de comunicación en MODO2

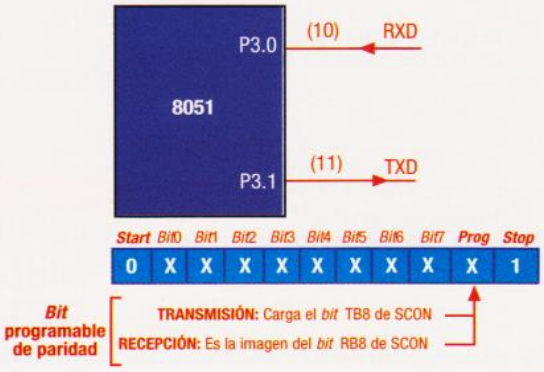


Figura 32.9 Esquema de comunicación y formato de palabra en los MODOS 2 y 3

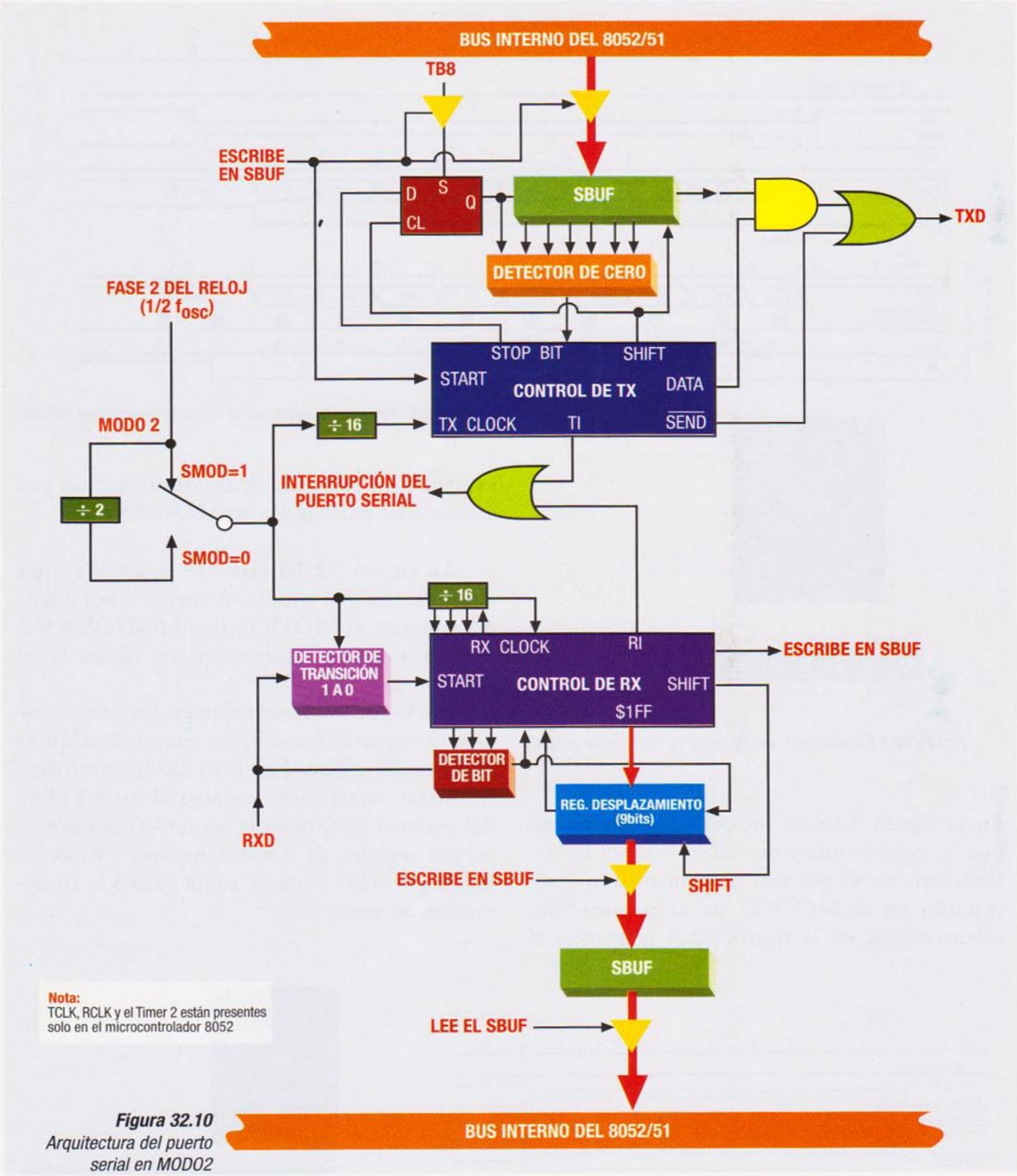


Figura 32.10
Arquitectura del puerto serial en MODO2

La recepción sin embargo, se inicia cuando se detecta un flanco o transición de nivel de voltaje descendente por el pin RXD, que es continuamente muestreado; y, al final del proceso de recepción se transfieren los 8 *bits* recibidos al registro SBUF, se carga el *bit* de parada en la posición de RB8, y se activa en 1 lógico la bandera de interrupción de recepción RI.

LOS MICROCONTROLADORES AVR DE 8 BITS DE ATMEL

Los microcontroladores AVR fabricados por ATMEL tienen arquitectura RISC y una estructura que soporta instrucciones compactas y poderosas de un solo ciclo.

Estos microcontroladores cuentan con una estructura para los periféricos internos (los puertos I/O, el oscilador interno, los *timers*, la UART, el SPI, los canales ADC y los comparadores análogos), que reduce la necesidad de usar componentes externos.

ATMEL

Es una compañía que desde 1984 es líder y pionera en el diseño y desarrollo de semiconductores avanzados con alta escala de integración; entre los principales productos de esta empresa se incluyen los microcontroladores y microprocesadores AVR, los conjuntos lógicos y analógicos programables de alta densidad (PAL y PAA), las memorias no volátiles, los mezcladores y amplificadores de señal, y los circuitos integrados de comunicación RF. Por su tecnología de fabricación de circuitos integrados con alta densidad (*wafer processes*), ATMEL es también considerada como una empresa de vanguardia; sus técnicas se orientan a las estructuras CMOS, BiCMOS y SiGe (*Silicon Germanium*).

En esta lección estudiaremos el microcontrolador AVR de 8 *bits* de ATMEL que cuenta con las características más representativas de los microcontroladores de nivel medio: el AT90S2313, que es un miembro de la familia AT90.

El microcontrolador AT90S2313 de la familia AT90

Este microcontrolador se considera de nivel medio y cuenta con una arquitectura avanzada propia de la familia AT90, la que posee muchas características importantes y útiles para el usuario a la hora de efectuar cualquier tarea. El AT90S2313 es un microcontrolador CMOS de 8 *bits* que cuenta con 2K*bytes* de memoria FLASH programable, y su estructura de empaque es tipo PDIP o SOIC de 20 *pines*. Sus principales características son:

- Arquitectura RISC de bajo consumo, propia de los MCU AVR
- 118 instrucciones compactas y poderosas
- 32 registros de 8 *bits* de propósito general
- Máxima velocidad de ejecución 10MHz (10MIPS)
- Memorias de datos y programa no volátiles
- 2K*bytes* de memoria programable FLASH
- 128 *bytes* de memoria RAM estática SRAM
- 128 *bytes* de memoria programable EEPROM
- Seguro de programación para las memorias FLASH y EEPROM
- 15 líneas I/O programables
- *Timer/Counters* de 8 y 16 *bits* con preescalador individual
- Comparador análogo
- *Watcthdog* programable
- Interfaz serial SPI
- UART *full-duplex*
- Dos modos de bajo consumo: *IDLE* y *POWER-DOWN*
- Interrupciones externas e internas
- Voltaje de alimentación entre 2,7 y 6,0V

Descripción de pines del AT90S2313

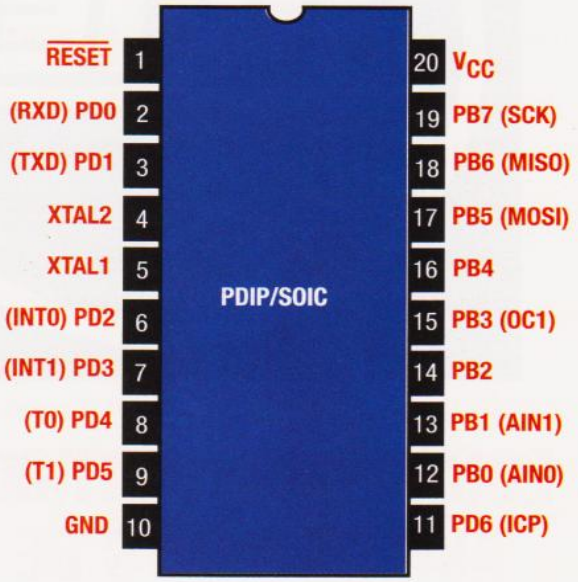


Figura 33.1 *Configuración de pines del microcontrolador AT90S2313*

En la **figura 33.1** se muestra la distribución y el nombre principal y alterno, entre paréntesis, de cada uno de sus *pines*. Muchos de sus *pines* tienen funciones ya bastante conocidas, y, por lo tanto, sólo se explicarán a continuación aquellos que poseen características particulares y nuevas que deban conocerse.

Port B (PB7- PB0): estos pines conforman el puerto B, que es de 8 *bits*, bidireccional (operan como entrada o salida), y todas sus líneas tienen además resistencias internas *pull-up* programables; todas sus líneas pueden manejar directamente diodos

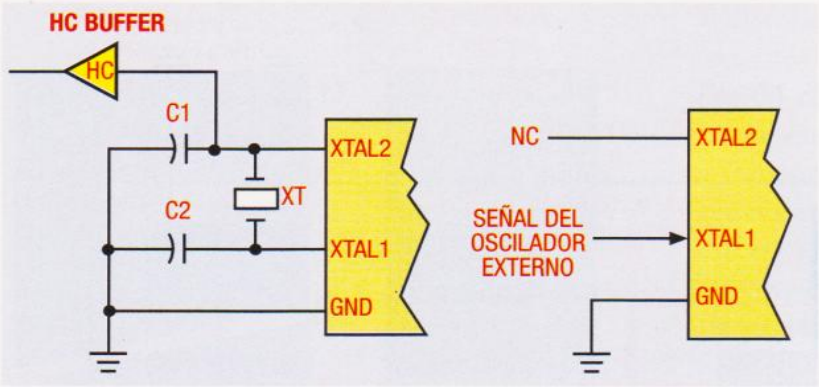


Figura 33.2 *Configuración del oscilador*

LED y cargas hasta de 20mA. PB0 y PB1 operan también como las entradas positiva (**AIN0**) y negativa (**AIN1**) del comparador análogo.

Al activar el *reset* todas sus líneas se ponen en tercer estado (alta impedancia).

Port D (PD6- PD0): estos *pines* son el puerto D de 7 *bits*, bidireccionales, y todas sus líneas tienen resistencias internas de *pull-up*; además, tienen capacidad para manejar hasta 20mA por cada una y su estado, después del *reset,* es el de alta impedancia.

$\overline{\text{RESET}}$: el *reset* en estos microcontroladores se activa por medio de este pin, si se pone externamente en nivel lógico bajo al menos durante 50ns.

Oscilador: en este módulo, XTAL1 y XTAL2 son los *pines* de la entrada y la salida, respectivamente, del amplificador inversor del oscilador interno. Tal como se muestra en la **figura 33.2**, el oscilador puede excitarse usando un resonador cerámico o un cristal de cuarzo, y, además, puede usarse para sincronizar otros dispositivos si se conecta un *buffer* en el *pin* XTAL2; allí también puede verse que la entrada para el oscilador externo, cuando se utiliza, debe ser el *pin* XTAL1.

Arquitectura del AT90S2313

La arquitectura AVR es *Harvard* (con buses separados para las memorias de datos y programa) y del tipo RISC. Las operaciones con la ALU, que pueden ser de tipo lógico, aritmético o de *bits*, se desarrollan (se ejecutan y se devuelve el resultado) en un sólo ciclo de instrucción y operan directamente sobre los 32 registros de trabajo. Por su estructura RISC, cuenta con un reducido número de instrucciones flexibles y poderosas, y, adicionalmente, su tecnología para el manejo de dos operandos simultáneamente les otorga la posibilidad de acceder rápidamente a los datos a través del banco formado por los 32 registros de trabajo de 8 *bits* cada uno.

En la **figura 33.3** se muestra el diagrama esquemático en bloques de la arquitectura del AT90S2313.

Organización de la memoria

En la **figura 33.4** se muestra la distribución de la memoria en los microcontroladores AT90S2313 y las direcciones asignadas para cada área. El espacio de memoria en la arquitectura AVR es lineal y está conformado por: 2K*bytes* de memoria de programa tipo FLASH, 128 *bytes* de memoria de datos EEPROM, 128 *bytes* de memoria de datos RAM estática (SRAM), 15 registros I/O de 1 *byte*, 32 registros de trabajo o propósito general, y los registros especiales que soportan las comunicaciones y los periféricos internos.

• Los registros de trabajo

Los 32 registros de trabajo de 8 *bits* de la CPU se encuentran localizados en las primeras posiciones del área de la memoria de datos (entre las posiciones $00 y $1F), y poseen la estructura que se muestra en la **figura 33.5**. Tal como se muestra, cada registro posee un nombre, una dirección específica, y, adicionalmente, los registros que hay entre el R26 y el R31 se pueden agrupar en tres registros de 16 *bits*: los registros X, Y y Z, que sirven como apuntadores en los modos de direccionamiento indirecto del espacio de memoria de datos; en la **figura 33.6** se muestran los registros X, Y y Z.

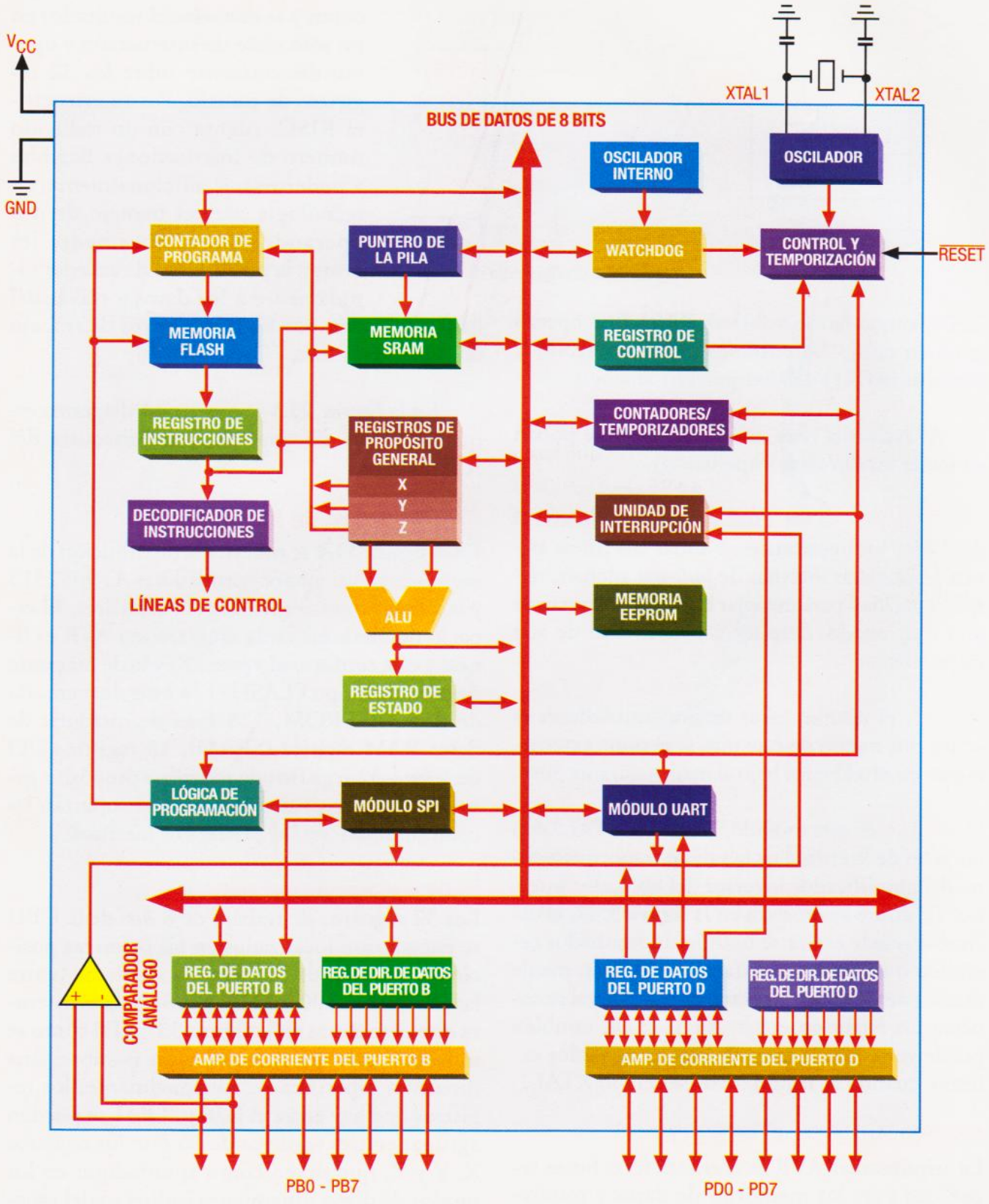


Figura 33.3 *Arquitectura del AT90S2313*

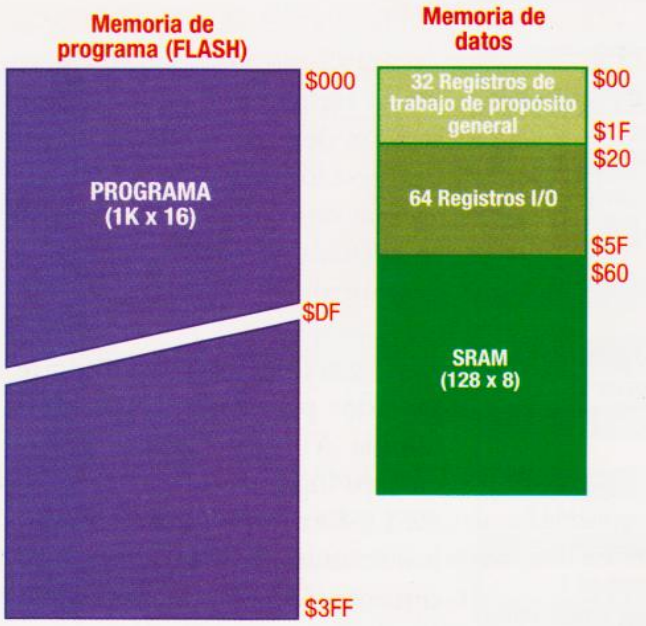


Figura 33.4 *Mapa de memoria del AT90S2313*

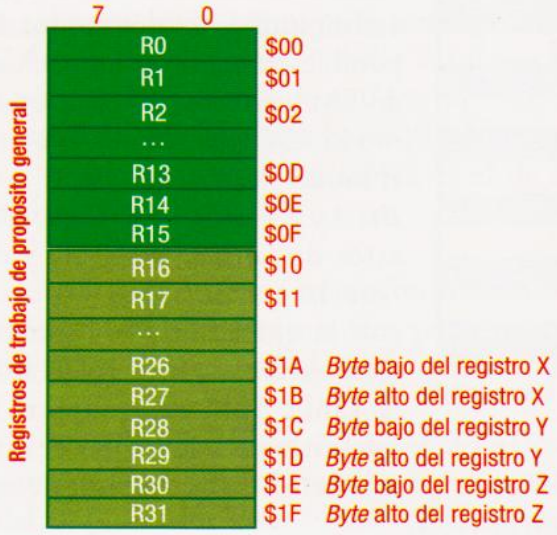


Figura 33.5 *Registros de trabajo de la CPU2313*

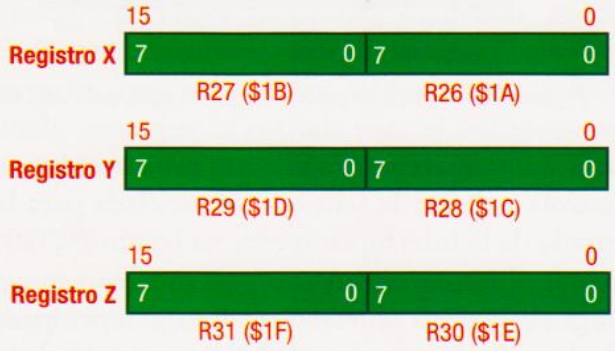


Figura 33.6 *Registros de direccionamiento indirecto X, Y y Z*

• El espacio de memoria I/O (el conjunto de registros especiales)

Las 64 direcciones I/O que conforman el espacio de memoria de datos de lectura y escritura, y a través de las cuales se controlan las funciones generales, especiales y el estado de los periféricos internos, tales como los *timers,* los contadores, los convertidores A/D, la UART y otros; se encuentran localizados entre las posiciones $20 y $5F. En la **tabla 33.1** se muestran las direcciones, los nombres y las funciones de todos los registros del espacio de memoria I/O.

Aunque todos los registros del espacio I/O son importantes y de manera particular cumplen funciones específicas y propias del control de cada uno de los módulos periféricos, hay dos registros que poseen características útiles y necesarias para el funcionamiento general del microcontrolador y que deben conocerse bien, éstos son: el registro de estado SREG y el de control de la CPU **MCUCR**.

1. Registro de estado SREG

Bit	7	6	5	4	3	2	1	0	
$3F ($5F)	I	T	H	S	V	N	Z	C	**SREG**
Lectura/Escritura	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

Figura 33.7 *Registro de estado SREG*

En la **figura 33.7** se muestra el registro de estado SREG con su dirección y el valor inicial que asume después del *reset*, en éste se encuentran los siguientes *bits* de control:

Bit **7 (I):** es el habilitador global de interrupciones, su estado debe ser uno lógico para habilitarlas y es borrado automáticamente por el *hardware* después de haber atendido la interrupción.

Bit **6 (T):** es el *bit* de almacenamiento y copia que se considera como fuente o destino para las instrucciones que operan sobre un *bit*, tales como BLD (carga de un *bit*) y BST (almacenamiento de un *bit*).

Bit **5 (H):** es el *bit* bandera que señala la ocurrencia del acarreo intermedio en las operaciones aritméticas.

DIRECCIÓN HEX.	NOMBRE	FUNCIÓN
$3F ($5F)	SREG	Registro de estado
$3D ($5D)	SPL	Puntero bajo de la pila
$3B ($5B)	GIMSK	Registro general de interrupción MASK
$3A ($5A)	GIFR	Registro general de interrupción FLAG
$39 ($59)	TIMSK	Registro de interrupción del TIMER/COUNTER
$38 ($58)	TIFR	Registro de bandera del TIMER/COUNTER
$35 ($55)	MCUCR	Registro de control general del MCU
$33 ($53)	TCCR0	Registro de control del TIMER/COUNTER 0
$32 ($52)	TCNT0	TIMER/COUNTER 0 (8 *bits*)
$2F ($4F)	TCCR1A	Registro de control A del TIMER/COUNTER 1
$2E ($4E)	TCCR1B	Registro de control B del TIMER/COUNTER 1
$2D ($4D)	TCNT1H	*Byte* alto del TIMER/COUNTER 1
$2C ($4C)	TCNT1L	*Byte* bajo del TIMER/COUNTER 1
$2B ($4B)	OCR1AH	*Byte* alto del registro de salida de comparación 1
$2A ($4A)	OCR1AL	*Byte* bajo del registro de salida de comparación 1
$25 ($45)	ICR1H	*Byte* alto del registro de entrada de captura 1
$24 ($44)	ICR1L	*Byte* bajo del registro de entrada de captura 1
$21 ($41)	WDTCR	Registro de control del temporizador del *watchdog*
$1E ($3E)	EEAR	Registro de direcciones de la EEPROM
$1D ($3D)	EEDR	Registro de memoria de la EEPROM
$1C ($3C)	EECR	Registro de control de la EEPROM
$18 ($38)	PORTB	Registro de datos del puerto B
$17 ($37)	DDRB	Registro de dirección de datos del puerto B
$16 ($36)	PINB	Pines de entrada del puerto B
$12 ($32)	PORTD	Registro de datos del puerto D
$11 ($31)	DDRD	Registro de dirección de datos del puerto D
$10 ($30)	PIND	Pines de entrada del puerto D
$0C ($2C)	UDR	Registro de datos I/O de la UART
$0B ($2B)	USR	Registro de estado de la UART
$0A ($2A)	UCR	Registro de control de la UART
$09 ($29)	UBRR	Registro de velocidad de la UART
$08 ($28)	ACSR	Registro de estado y control del comparador análogo

Tabla 33.1 *Registros especiales del espacio de memoria I/O*

***Bit* 4 (S):** el *bit* de signo señala con su estado, en uno lógico, cuando el resultado de una operación aritmética, en complemento a dos, es negativo.

***Bit* 3 (V):** la bandera de sobreflujo o desbordamiento se genera cuando ocurre un sobrepaso en el alcance del resultado de una operación en complemento a dos.

***Bit* 2 (N):** la bandera de resultado negativo señala cuando una operación aritmética cualquiera, o lógica, arroja un resultado negativo.

***Bit* 1 (Z):** este *bit* es la bandera que nos indica cuando el resultado de una operación lógica o aritmética es nulo o cero.

***Bit* 0 (C):** este *bit* es la bandera de acarreo final propia de las operaciones aritméticas y lógicas.

2. Registro de control de la CPU MCUCR

En la **figura 33.8** se muestra el registro de control MCUCR con su dirección y el valor inicial que adopta después del *reset*; en éste se encuentran los siguientes *bits* de control:

***Bit* 7 y 6 (R):** estos *bits* están reservados para otros MCU de la familia AT90 y siempre se leen en cero lógico.

***Bit* 5 (SE):** es el *bit* que se utiliza para habilitar cualquier modo de adormecimiento (*SLEEP*), con su estado en uno lógico está habilitado.

***Bit* 4 (SM):** es el *bit* que se usa como interruptor de control para activar uno de los dos modos disponibles de bajo consumo. Cuando SM vale cero, se selecciona el modo *IDLE*, y cuando vale uno, el modo *POWER-DOWN*.

***Bit* 3 y 2 (ISC11 e ISC10):** con estos dos *bits* se controla y define la sensibilidad asociada con la entrada de la interrupción externa 1 (pin INT1). En la **tabla 33.2** se resumen las opciones de selección del nivel y el flanco para la señal de excitación de esta interrupción.

Bit	7	6	5	4	3	2	1	0	
$35 ($55)	-	-	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
Lectura/Escritura	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

Figura 33.8 *Registro de control del microcontrolador MCUCR*

Bit 1 y 0 (ISC01 e ISC00): con estos dos *bits* se controla y define la sensibilidad asociada para la entrada de la interrupción externa 0 (pin INT0).

En la **tabla 33.3** se muestran las opciones para la selección del nivel y el flanco de la señal de excitación de tal interrupción.

ISC11	ISC10	DESCRIPCIÓN
0	0	Se genera una interrupción por nivel bajo en el pin INT1
0	1	Reservado
1	0	Se genera una interrupción por un flanco descendente en el pin INT1
1	1	Se genera una interrupción por un flanco ascendente en el pin INT1

Tabla 33.2 *Control de sensibilidad de la interrupción 1 (INT1)*

ISC01	ISC00	DESCRIPCIÓN
0	0	Se genera una interrupción por nivel bajo en el pin INT0
0	1	Reservado
1	0	Se genera una interrupción por un flanco descendente en el pin INT0
1	1	Se genera una interrupción por un flanco ascendente en el pin INT0

Tabla 33.3 *Control de sensibilidad de la interrupción 0 (INT0)*

• La memoria de programa FLASH del sistema

Este microcontrolador ofrece 2K*bytes* de memoria FLASH programable por el usuario en donde se debe almacenar el programa, y está organizada en una estructura de 1K X16 para facilitar las operaciones con las instrucciones de 16 ó 32 *bits*. Por medio del registro contador del programa (PC), que es de 10 *bits*, se pueden direccionar hasta 1.024 posiciones de la memoria de programa.

Figura 33.9 *Área de memoria SRAM*

• La memoria de datos SRAM

El área de la memoria de datos es del tipo RAM estática (SRAM) y está compuesta por 128 *bytes* de 8 *bits* localizados entre las posiciones $60 y $DF; en toda esta área, los registros se pueden acceder directamente por medio de cualquier modo de direccionamiento. En la **figura 33.9** se muestra la organización de la SRAM.

Durante las interrupciones y los saltos a subrutinas, la dirección de retorno del PC es almacenada en el área de la pila (*stack*), que puede estar localizada en cualquier lugar de la memoria SRAM, y, por lo tanto, se puede afirmar que su tamaño sólo está limitado por el espacio disponible en la memoria de datos estática. El registro apuntador de la pila SP (*stack pointer*) es un registro de 8 *bits* que se usa para direccionar todos los 128 *bytes* de la SRAM; éste se decrementa en uno cada vez que se apila un nuevo dato por medio de la instrucción PUSH, y, por el contrario, se incrementa en uno cuando un dato es descargado de la pila por medio de la instrucción POP; en la **figura 33.10** se puede apreciar la parte baja del registro SP, llamada SPL.

Bit	7	6	5	4	3	2	1	0	
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	**SPL**
Lectura/Escritura	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

Figura 33.10 *Registro apuntador de la pila SPL*

• La memoria de datos EEPROM

El MCU tiene internamente una memoria de datos de lectura y escritura de 128 *bytes* del tipo EEPROM y está localizada en un espacio separado dentro del área de datos. A continuación, en la **figura 33.11**, se muestran los registros propios del manejo de datos de esta memoria, allí se pueden ver: el registro de dirección EEAR (EEPROM *address register*) que contiene la dirección de la localidad de memoria que va a ser

accedida, el registro de datos **EEDR** (EEPROM *data register*) que contiene el dato de 8 *bits* que va a ser escrito o leído de la memoria, y el registro de control **EECR** (EEPROM *control register*); éste a su vez tiene tres *bits* de control: el **EEMWE**, el **EEWE** y el **EERE**, los cuales son utilizados como el habilitador maestro de la escritura, el habilitador de escritura y el habilitador de lectura, respectivamente.

Bit	7	6	5	4	3	2	1	0	
$1E ($3E)	-	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	**EEAR**
Lectura/Escritura	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
$1D ($3D)	MSB							LSB	**EEDR**
Lectura/Escritura	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
$1C ($3C)	-	-	-	-	-	EEMWE	EEWE	EERE	**EECR**
Lectura/Escritura	R	R	R	R	R	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

Figura 33.11 *Registros de control y estado de la memoria EEPROM*

Nota:
la memoria FLASH interna de programa del AT90S2313 soporta hasta 1.000 ciclos de lectura y escritura, lo cual es típico para esta tecnología. Sin embargo, las memorias EEPROM tienen una mayor duración, pueden soportar hasta 100.000 ciclos.

Modos de direccionamiento de datos y programas

El microcontrolador AT90S2313, con arquitectura AVR RISC, cuenta con un poderoso conjunto de once modos de direccionamiento que nos permiten acceder a cualquiera de sus memorias y registros internos eficientemente. A continuación se hace la descripción y el análisis simplificado de cada uno de estos modos.

1. Direccionamiento directo de registro simple

En la **figura 33.12**, **OP** representa al código de operación de la instrucción y el registro **d** representa al operando.

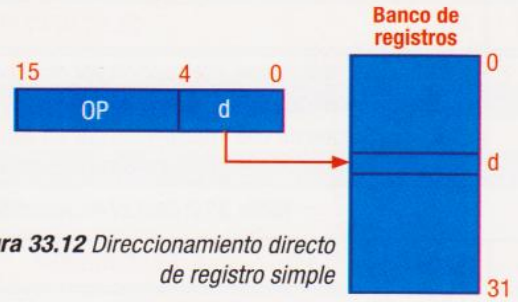


Figura 33.12 *Direccionamiento directo de registro simple*

2. Direccionamiento directo de registro doble

En la **figura 33.13**, **OP** representa al código de operación de la instrucción y los registros r y d representan a los operandos. El resultado se guarda en el registro **d**.

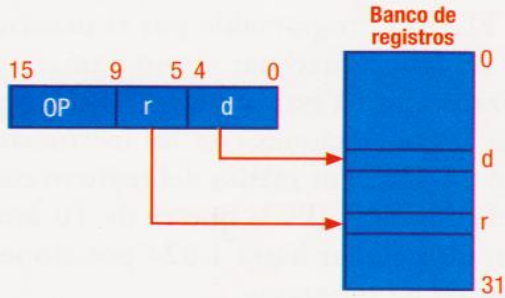


Figura 33.13 *Direccionamiento directo de registro doble*

3. Direccionamiento directo de registros I/O

En la **figura 33.14**, **OP** representa al código de operación de la instrucción y la dirección del operando está contenida en los 6 *bits* más bajos de la instrucción **P**; mientras que en n se ubica el destino o la fuente del registro direccionado.

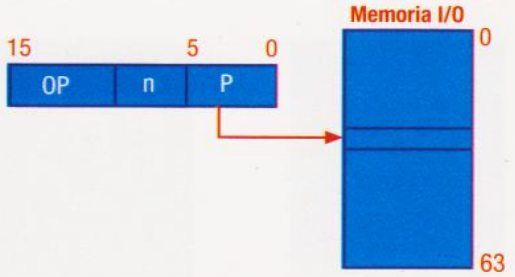


Figura 33.14 *Direccionamiento directo de registros I/O*

4. Direccionamiento directo de datos

En la **figura 33.15**, **OP** representa al código de operación de la instrucción y la dirección de 16 *bits* del dato se localiza en la parte baja de la instrucción. **Rr** o **Rd** especifican el registro destino o fuente.

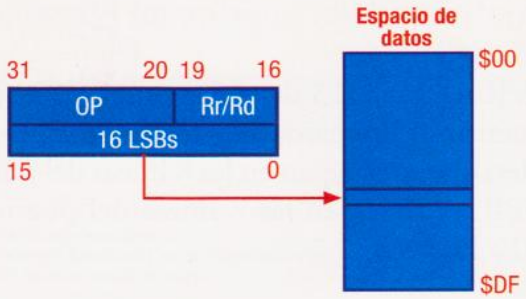


Figura 33.15 *Direccionamiento directo de datos*

5. Direccionamiento indirecto de datos con desplazamiento

En la **figura 33.16**, **OP** representa al código de operación de la instrucción y la dirección del operando se obtiene al sumar el contenido del registro **Y** o **Z**, con la dirección de 6 *bits* (desplazamiento) contenida en la parte a de la instrucción. En n se ubica el destino o la fuente del registro direccionado.

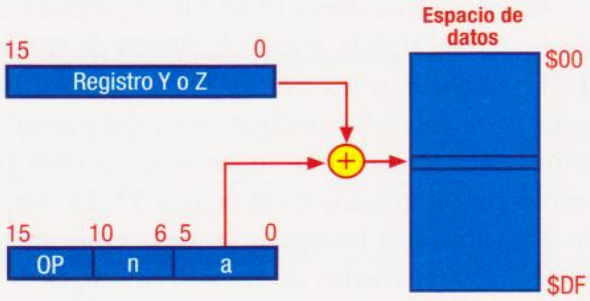


Figura 33.16 *Direccionamiento indirecto de datos con desplazamiento*

6. Direccionamiento indirecto de datos

En la **figura 33.17**, la dirección del operando se encuentra en cualquiera de los registros apuntadores **X**, **Y**, o **Z**.

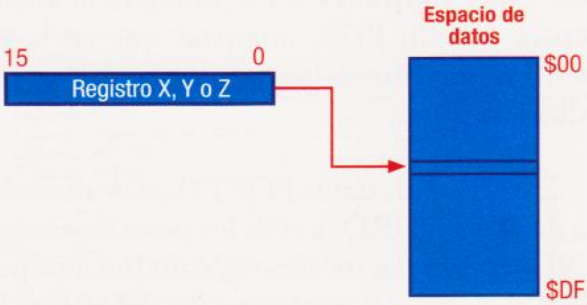


Figura 33.17 *Direccionamiento indirecto de datos*

7. Direccionamiento indirecto de datos con pre-decremento

En la **figura 33.18**, el contenido del registro **X**, **Y**, o **Z** se disminuye en uno, antes de la operación para apuntar a la dirección efectiva del operando.

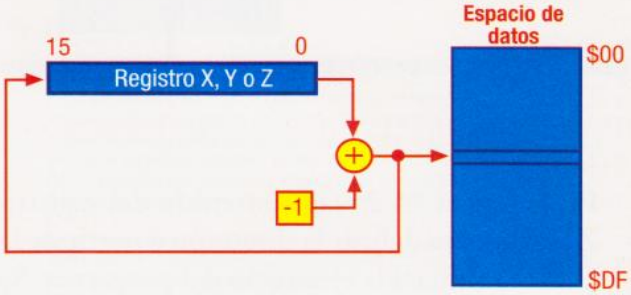


Figura 33.18 *Direccionamiento indirecto de datos con pre-decremento*

8. Direccionamiento indirecto de datos con post-incremento

En la **figura 33.19**, el contenido del registro **X**, **Y**, o **Z** se incrementa en uno después de la operación; de esta forma el contenido de cualquiera de estos registros, antes del incremento, apunta a la dirección efectiva del operando.

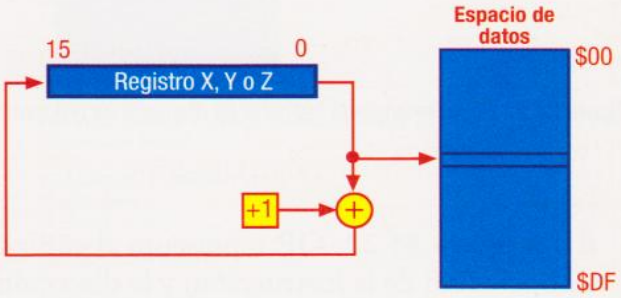


Figura 33.19 *Direccionamiento indirecto de datos con post-incremento*

9. Direccionamiento de código de memoria con constante

En la **figura 33.20**, el contenido del registro Z se usa como un *byte* constante de dirección. En los 15 *bits* más significativos se encuentra la dirección efectiva (entre las posiciones 0 y 1k) y con el *bit* menos significativo (*bit*0) se selecciona la opción de captura de la parte baja o la alta del *byte* del código de memoria, al ponerse en 0 y 1 respectivamente.

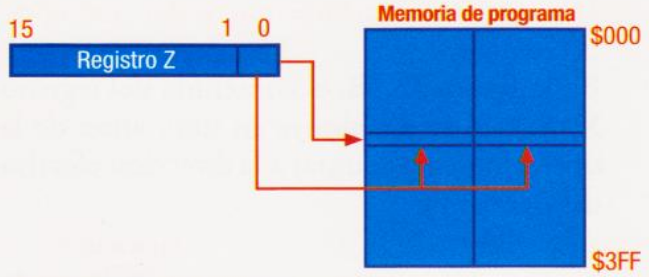


Figura 33.20 *Direccionamiento de código de memoria con constante*

10. Direccionamiento indirecto de memoria de programa

En la **figura 33.21**, el contenido del registro **Z** se usa para definir la dirección a partir de la cual continuará la ejecución del programa. Se usa con las instrucciones de salto y llamado (IJMP e ICALL), y, en cualquier caso, el registro contador del programa PC se carga con el contenido de Z.

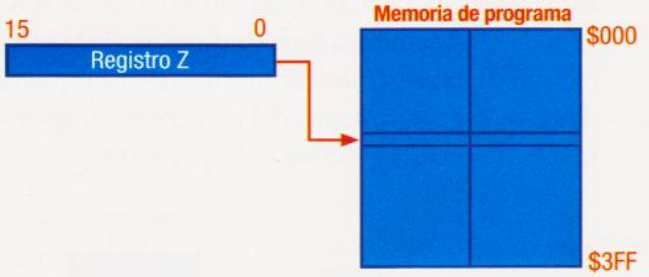


Figura 33.21 *Direccionamiento indirecto de memoria de programa*

11. Direccionamiento relativo de memoria de programa

En la **figura 33.22**, **OP** representa al código de operación de la instrucción, y la dirección relativa en complemento a dos **k** puede estar entre -2.048 y 2.047. Al ejecutarse el programa salta y continúa corriendo a partir de la dirección efectiva que resulta de sumar el contenido del registro PC con la dirección relativa más uno (**dirección efectiva=PC+k+1**).

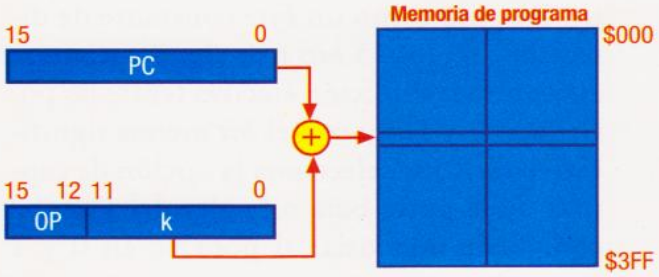


Figura 33.22 *Direccionamiento relativo de memoria de programa*

Estructura de los puertos de entrada y salida I/O

El AVR AT90S2313 tiene disponibles 15 líneas de puertos bidireccionales y totalmente programables, que se agrupan en las 8 líneas del puerto B (PB0 a PB7) y en las 7 líneas del puerto D (PD0 a PD6).

1. El puerto B

PB es un puerto I/O bidireccional de 8 *bits*, todos tienen resistencia interna de *pull-up* programable individualmente, y, además, cuentan con *buffers* con corriente limitada capaces de manejar directamente diodos LED y cargas de hasta 20mA en configuración *sink*. En la **figura 33.23** se muestra la arquitectura del pin PB0, mientras que en la **tabla 33.4** se resumen las funciones alternas de algunos de sus *pines*.

El registro de datos PORTB, el de direcciones de datos DDRB, y el de los pines de entrada PINB conforman los tres registros que nos permiten el acceso y la configuración del puerto B; todos hacen parte de los registros especiales I/O y están representados en la **figura 33.24**. La **tabla 33.5** muestra las opciones de configuración del puerto por medio de los *bits* del registro de dirección DDRB (*bits* DDBx)

2. El puerto D

PD es un puerto I/O bidireccional de 7 *bits*, todos tienen resistencia interna de *pull-up* programable individualmente y, además, cuentan con *buffers* que tienen la capacidad de impulsar hasta 20mA en configuración *sink*. En la **figura 33.25** se muestra la arquitectura del pin PD0, mientras que en la **tabla 33.6** se resumen las funciones alternas de todos sus pines.

El registro de datos PORTD, el de direcciones de datos DDRD, y el de los pines de entrada PIND conforman los tres registros que nos permiten el acceso y la configuración del puerto D. Éstos están representados en la **figura 33.26**.

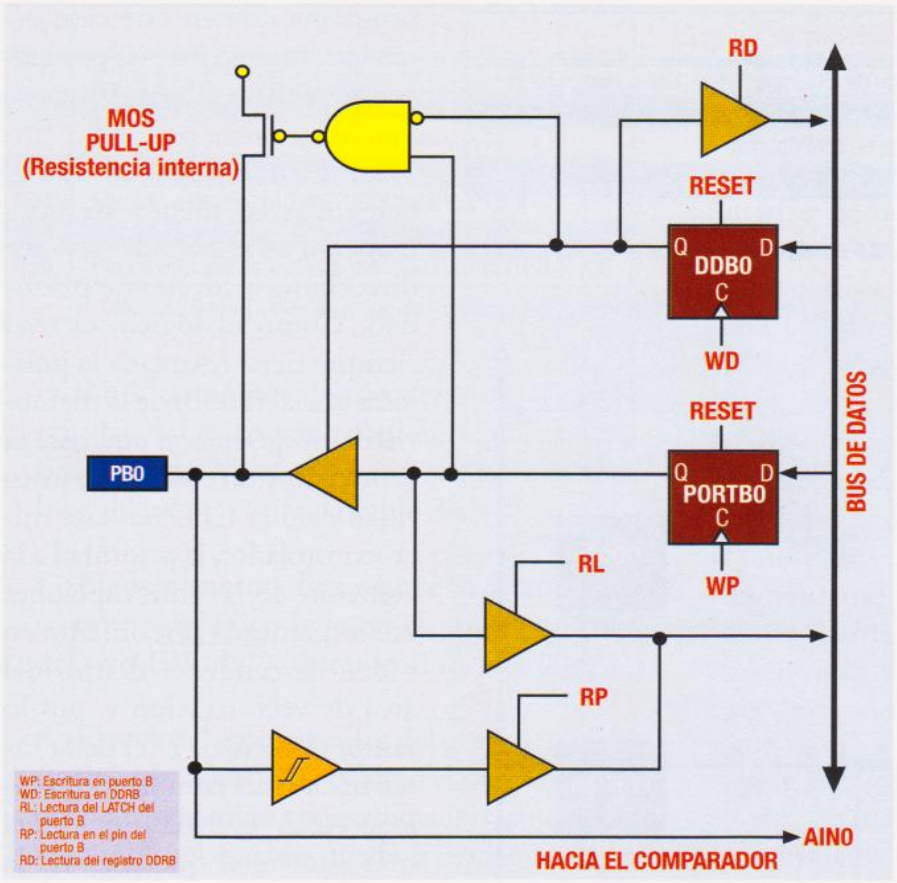


Figura 33.23 *Arquitectura simplificada de un pin del puerto B*

PIN	FUNCIONES ALTERNAS
PB0	Entrada positiva del comparador análogo AIN0
PB1	Entrada negativa del comparador análogo AIN1
PB3	Salida de comparación e igualación del TIMER/COUNTER 1, OC1
PB5	Línea de entrada de datos para descarga de la memoria, MOSI
PB6	Línea de salida de datos para la carga de la memoria, MISO
PB7	Entrada serial de reloj SCK

Tabla 33.4 *Funciones alternas de algunos pines del puerto B*

Bit	7	6	5	4	3	2	1	0	
$18 ($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	**PORTB**
Lectura/Escritura	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
$17 ($37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	**DDRB**
Lectura/Escritura	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
$16 ($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	**PINB**
Lectura/Escritura	R	R	R	R	R	R	R	R	
Valor inicial	X	X	X	X	X	X	X	X	

Figura 33.24 *Registros de acceso y configuración del puerto B*

PARA SABER MÁS:
Dos aspectos importantes de los puertos

- Debido a sus arquitecturas, cuando los pines del puerto B o D se configuran como entradas y son puestos externamente en cero, pueden suministrar corriente a través de las resistencias internas de *pull-up* cuando éstas permanecen activas. **¡No olvide desconectarlas!**.

- Por medio de los registros de los *pines* de entrada PINB y PIND, se puede leer directamente el valor lógico presente en todo momento en cualesquiera de los pines de estos puertos, si están configurados como entradas. Mientras que cuando se leen los registros PORTB o PORTD, se accede a los *latchs* de datos de cada uno de los *pines*.

Las opciones de configuración del puerto se programan por medio de los *bits* del registro de dirección DDRD (*bits* DDDx), de la misma forma que se hace para el puerto B (ver la tabla 33.5).

El reset y las interrupciones

El AVR AT90S2313 nos ofrece 11 fuentes de interrupción, incluyendo al *reset*. Todos los vectores de las interrupciones apuntan, como en todos los microcontroladores, a las primeras posiciones de la memoria de

DDBx	PORTBx	I/O	Pull-up	Comentarios
0	0	Entrada	No	Estado de alta impedancia (tercer estado Hi-Z)
0	1	Entrada	Si	Con un cero lógico externo suministrará corriente
1	0	Salida	No	Salida en cero (toggle)
1	1	Salida	No	Salida en uno (toggle)

Nota: x puede ser uno de los pines entre el 0 y 7

Tabla 33.5 *Configuración de los pines del puerto B*

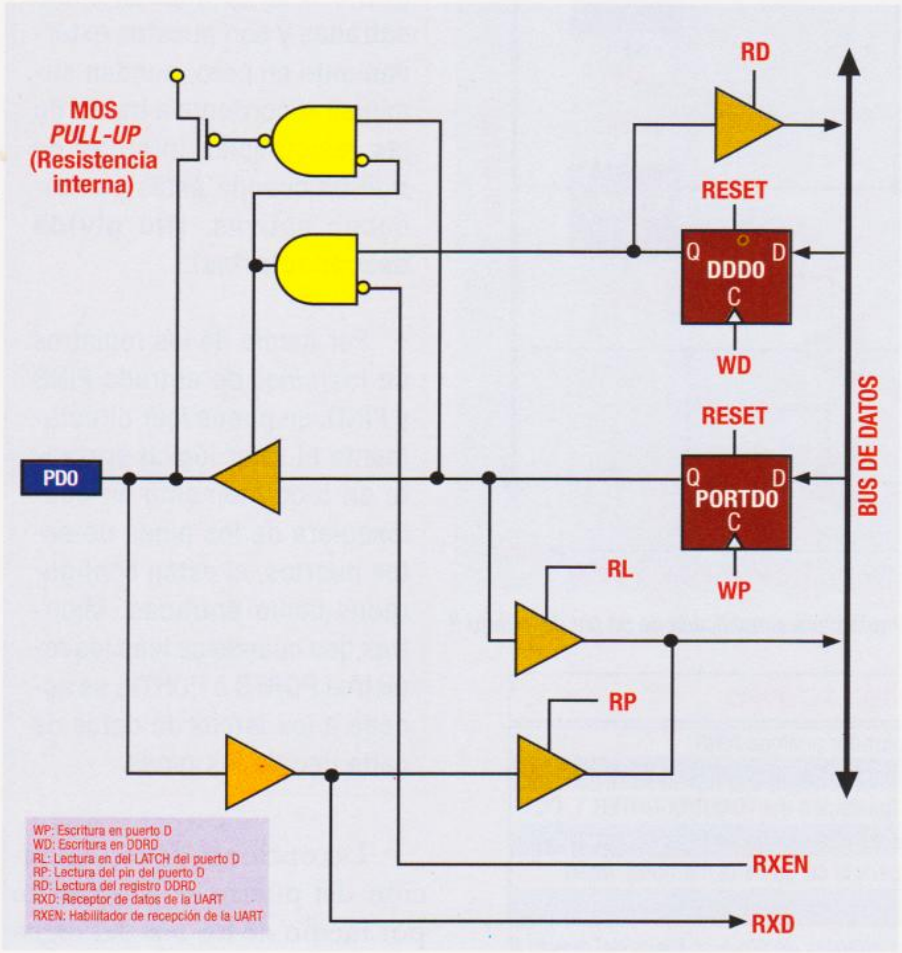


Figura 33.25 *Arquitectura simplificada de un pin del puerto D*

Bit	7	6	5	4	3	2	1	0	
$12 ($32)	-	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	**PORTD**
Lectura/Escritura	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
$11 ($31)	-	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	**DDRD**
Lectura/Escritura	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
$10 ($30)	-	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	**PIND**
Lectura/Escritura	R	R	R	R	R	R	R	R	
Valor inicial	0	X	X	X	X	X	X	X	

Figura 33.26 *Registros de acceso y configuración del puerto D*

programa que en este caso están localizados entre las posiciones $000(0) y $00A(10).

En la **tabla 33.7** se describen todas las fuentes de interrupción acompañadas por sus direcciones y niveles de prioridad. Como es lógico, el *reset* siempre tiene reservada la posición inicial ($000) de la memoria de programas y, además, es la interrupción con mayor prioridad para la CPU; en este microcontrolador, la prioridad a la atención de las interrupciones está organizada linealmente en orden descendente dentro del área de vectorización y, por lo tanto, el vector 11 (el de la interrupción del comparador análogo ANA_COMP), que es el más alto, es el que goza de la menor prioridad.

Todas las fuentes de interrupción de la CPU pueden ser individualmente habilitadas o no por medio de los *bits* de los registros de control adecuados para cada una de ellas; pero, en cualquier caso, no se debe olvidar poner un uno lógico en el *bit* habilitador global de interrupciones (I) que se encuentra en el registro de estado **SREG**.

1. El reset

Las fuentes de interrupción que generan un ciclo de *reset* en el AVR AT90S2313 son internas y externas, y son tres:

- **El *reset* a la conexión (*power-on-reset*).** Éste ocurre cuando la fuente de alimen-

PIN	FUNCIONES ALTERNAS
PD0	Entrada de recepción de datos de la UART, RXD
PD1	Salida de transmisión de datos de la UART, TXD
PD2	Entrada de interrupción externa cero, INT0
PD3	Entrada de interrupción externa uno, INT1
PD4	Entrada externa del TIMER/COUNTER 0, T0
PD5	Entrada externa del TIMER/COUNTER 1, T1
PD6	Pin de entrada de captura del TIMER/COUNTER 1, ICP

Tabla 33.6 *Funciones alternas de los pines del puerto D*

tación Vcc cae por debajo del umbral de seguridad del microcontrolador, el cual está definido como **Vpot**, y para el que se muestran los valores típicos en la **tabla 33.8**.

- **El reset externo.** Éste ocurre cuando se aplica un nivel lógico bajo en el pin de entrada del *reset* ($\overline{\text{RESET}}$), durante al menos 50ns.

- **El *reset* del temporizador del perro-guardián (*watchdog*).** Éste se da cuando el temporizador del perro-guardián está habilitado y su tiempo se cumple; el usuario puede seleccionar el uso de este temporizador si lo programa utilizando el fusible lógico de programación de la memoria FLASH, llamado **FSTRT**.

Nota: después del *reset*, todos los registros especiales I/O son puestos en su valor inicial, y el contador del programa (PC) se carga con la dirección de origen ($000).

2. Las interrupciones externas

Las interrupciones externas son dos y se reciben a través de los *pines* **INT1** e **INT0**, las dos se habilitan por medio de un flanco externo creciente o decreciente, ó, por medio de un nivel lógico bajo; estas condiciones se programan por medio del registro de control del microcontrolador **MCUCR**, tal como se muestra en las tablas 33.2 y 33.3.

Como ya se ha dicho, la habilitación global de las interrupciones se efectúa con el ***bit* I** del registro de estado, pero la habilitación de cada interrupción externa de manera individual se logra por medio de los registros **GIMSK** y **GIFR**, que reciben los nombres de: máscara de interrupciones generales y banderas de interrupciones generales. A continuación se describen estos dos registros.

El registro máscara de interrupciones generales GIMSK

En este registro se encuentran habilitados para el usuario solamente los *bits* 6 y 7, denominados **INT0** e **INT1**, y, por medio de ellos, se habilitan las interrupciones externas 0 y 1 respectivamente (asociadas con las entradas INT0 e INT1), cuando se colocan en uno lógico. En la **figura 33.27** se muestra el registro GIMSK.

Vector Nº	Direcciones de Memoria de programa	Fuente de interrupción	Definición de las interrupciones	Prioridad
1	$000	RESET	Pin externo de reset, del reset a la conexión (POR) y del *watchdog*	Alta
2	$001	INT0	Solicitud de interrupción externa cero (pin INT0)	
3	$002	INT1	Solicitud de interrupción externa uno (pin INT1)	
4	$003	TIMER1 CAPT1	Evento de captura del TIMER/COUNTER 1	
5	$004	TIMER1 COMP1	Evento de igualación con el TIMER/COUNTER 1	
6	$005	TIMER1 OVF1	Sobrepasamiento del registro del TIMER/COUNTER 1	
7	$006	TIMER0 OVF0	Sobrepasamiento del registro del TIMER/COUNTER 0	
8	$007	UART, RX	Recepción completa (RX) de la UART	
9	$008	UART, UDRE	Registro de datos de la UART vacío	
10	$009	UART, TX	Transmisión completa (TX) de la UART	
11	$00A	ANA_COMP	Comparación análoga	Baja

Tabla 33.7 *Tabla de vectorización del AVR AT90S2313*

Símbolo	Parámetro	Min	Típico	Max	Unidades
V_{POT}	Voltaje ascendente del umbral del reset a la conexión	1,0	1,4	1,8	V
	Voltaje descendente del umbral del reset a la conexión	0,4	0,6	0,8	V
V_{RST}	Umbral de voltaje del pin externo de reset ($\overline{RESET}$)	-	-	$0{,}85V_{CC}$	V
t_{TOUT}	Periodo de retraso en el tiempo de salida del reset, cuando el fusible FSTRT no se programa	11,0	16,0	21,0	ms
t_{TOUT}	Periodo de retraso en el tiempo de salida del reset, cuando el fusible FSTRT se programa	0,25	0,28	0,31	ms

Nota: Todas las caractreísticas corresponden con un voltaje de alimentación V_{CC} = 5,0V

Tabla 33.8 *Niveles de seguridad del reset a la conexión y parámetros del reset*

Bit	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1	INT0	-	-	-	-	-	-	GIMSK
Lectura/Escritura	R/W	R/W	R	R	R	R	R	R	
Valor inicial	0	0	0	0	0	0	0	0	

Figura 33.27 *El registro máscara de interrupciones generales GIMSK*

El registro bandera de interrupciones generales GIFR

En este registro se encuentran habilitados para el usuario únicamente los *bits* 6 y 7, que se denominan las banderas de las interrupciones externas INTF0 e INTF1, y, por medio de ellos, se informa el estado de las solicitudes de las interrupciones externas 0 y 1, respectivamente. Estos dos *bits* se borran automáticamente cada vez que la rutina de la interrupción adecuada se ejecuta (luego de la instrucción de retorno de una interrupción RETI); y, alternativamente, pueden ser borrados por medio del *software* al escribir directamente un cero en ellos. En la **figura 33.28** se muestra el registro GIFR y sus *bits* bandera.

Bit	7	6	5	4	3	2	1	0	
$3A ($5A)	INTF1	INTF0	-	-	-	-	-	-	GIFR
Lectura/Escritura	R/W	R/W	R	R	R	R	R	R	
Valor inicial	0	0	0	0	0	0	0	0	

Figura 33.28 *El registro bandera de interrupciones generales GIFR*

Diagramas de tiempos de los estados de *reset*

En las **figuras 33.29** y **33.30** se muestran las condiciones propias de los dos estados de *reset* a la conexión POR (*power -on-reset*), que se dan cuando se inicia el microcontrolador por medio de la fuente de alimentación Vcc (*start -up- condition*) y también se muestra el diagrama de tiempos propío del *reset* externo cuando éste se ejecuta durante la operación normal de la CPU2313.

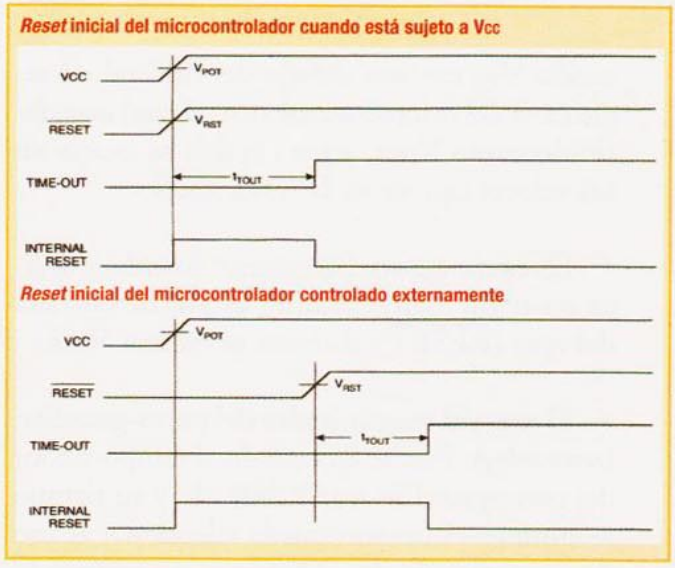


Figura 33.29 *Diagrama de tiempos del reset a la conexión (start-up) del AT90S2313*

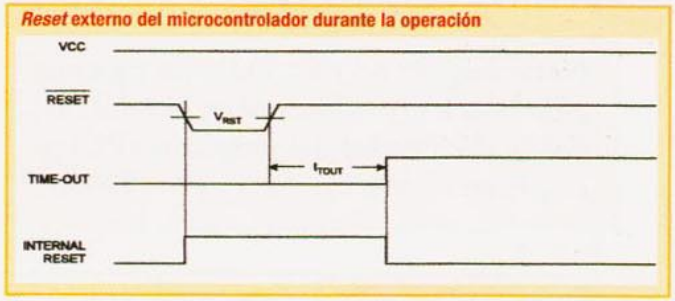


Figura 33.30 *Diagrama de tiempos del reset externo durante la operación del AT90S2313*

En la tabla 33.8 se resumen los parámetros más importantes y que conducen a un *reset* seguro; allí se muestran los valores típicos, mínimo y máximo del umbral del voltaje de disparo y los tiempos de respuesta que definen las condiciones del *reset*, tanto interno como externo. En el caso del *reset* externo, no se debe olvidar que el nivel VRST debe ser aplicado por 50ns como mínimo para que la CPU pueda validar su condición.

INDICE GENERAL

Introducción general ... 1
1 Introducción a los microcontroladores ... 3
Los microprocesadores ... 4
Evolución histórica de los microprocesadores ... 5
Los microprocesadores actuales ... 8
2 Los microprocesadores ... 9
El microprocesador, corazón de las computadoras ... 10
Funcionamiento del microprocesador ... 11
Organización interna de un microprocesador ... 13
Operación básica de un microprocesador ... 13
Estructura de bases de un microprocesador ... 15
El bus de direcciones. Espacio de memoria ... 16
El bus de control ... 17
Organización de un sistema basado en un microprocesador ... 17
Secuencia de operación de un sistema basado en un microprocesador ... 18
Secuencia de operación de una instrucción ... 22
Fase de búsqueda ... 23
Decodificación de una instrucción ... 24
Ejecución de una instrucción ... 24
Puertos de entrada y salida ... 25
El circuito de soporte 8155 (I/O-RAM) ... 26
Acceso a los puertos ... 27
Diseño de un sistema mínimo con 8085 y 8155 ... 28
Diseño de la tarjeta procesadora ... 28
El circuito de reloj ... 30
El circuito de reset ... 30
Los buses de datos y direcciones ... 30
El bus de control ... 31
Diseño de la tarjeta programadora ... 31
Notas finales ... 32
3 Los microcontroladores ... 33
Sistemas microcontrolados ... 35
Diferencias entre sistemas basados en microprocesador y en microcontrolador ... 37
Un microcontrolador para cada gusto ... 39
Principales familias de microcontroladores ... 40
La familia Intel 8051 ... 40
Motorola ... 40
Microchip ... 41
Atmel ... 42
4 Los microcontroladores PIC ... 43
Herramientas de desarrollo ... 44
Arquitectura de los PIC ... 44
El núcleo ... 44
Los periféricos ... 45
Características especiales ... 46
Variedad de los dispositivos ... 47
Variedades de memoria ... 47
EPROM ... 48
ROM ... 48
Memoria Flash ... 48
Opciones de rango de voltaje de operación ... 48
Variedades de encapsulado ... 48
Dispositivos borrables con luz ultravioleta ... 49
Dispositivos programables una vez (*One-Time-Programmable:* OTP) ... 49
Dispositivos FLASH ... 49
Dispositivos EEPROM ... 49
Dispositivos ROM ... 49
DIE ... 49
Servicios especiales ... 49
Programadores y entrenadores (*Trainers*) ... 50
5 Los microcontroladores Motorola ... 51
La familia HC05 ... 52
La familia HC08 ... 53
La familia HC11 ... 54
La familia HC12 ... 54
La familia HC16 ... 55
La familia Núcleo M (M CORE) ... 55
La familia MPC5XX ... 55
La familia 683XX ... 56
Sistemas de desarrollo ... 56
6 Los microcontroladores Intel, Atmel y Basic Stamp ... 57
Microcontroladores Intel ... 58
Núcleo 8051 ... 58
Microcontroladores Atmel ... 59
Basados en arquitectuta 8051 ... 60
AT'91 ARM Thumb ... 60
Microcontroladores AVR ... 60
Microcontroladores BASIC Stamp ... 60
El BASIC Stamp I ... 61
El BASIC Stamp II ... 62
7 El oscilador y el circuito de reinicialización (*reset*) en los microcontroladores PIC ... 65
El circuito oscilador ... 66
Osciladores basados en cristal o en resonadores cerámicos ... 67
Circuito de reinicialización (*reset*) ... 68
Reinicialización al encendido (*Power On Reset*: POR) ... 69
Temporizador al encendido (*Power Up Timer*: PWRT) ... 70
Temporizador de oscilación al encendido (OST) ... 70
Secuencia seguida al alimentar el sistema ... 71
Reinicialización por caida en la alimentación (*Brown-Out Reset*: BOR) ... 71
Registro de control de potencia (*Power Control*: PCON) ... 72
8 Arquitectura de los microcontroladores PIC ... 73
Arquitectura Harvard ... 74
Instrucciones de palabra sencilla (*Single word*) ... 74
Manejo de las instrucciones (*Pipeline*) ... 74
Conjunto reducido de instrucciones ... 75
Estructura interna dividida en registros ... 75
Instrucciones ortogonales (simétricas) ... 75
El ciclo de instrucción ... 76
Formato general de una instrucción ... 76
Reloj de las instrucciones ... 76
Conjunto (*set*) de instrucciones ... 77
La ALU y la CPU ... 77
Unidad aritmética lógica (ALU) ... 79
El registro de estado ... 79
El registro OPCION ... 80
9 La memoria en los microcontroladores PIC ... 81
El contador de programa ... 82
La memoria de programa ... 82
La memoria de datos ... 87
10 La EEPROM de datos en los microcontroladores PIC ... 91
Memorias EEPROM internas ... 92
Ciclo de lectura ... 93
Ciclo de escritura ... 94
Memorias EEPROM seriales ... 95
Ejemplo de aplicación ... 96
11 Interrupciones en los microcontroladores PIC ... 97
Interrupción externa ... 100
Ejemplo de interrupción externa ... 102
Interrupciones por cambio en un pin RB ... 102
Problemas comunes al trabajar con interrupciones ... 103
Comentarios finales ... 104
12 Los temporizadores en los microcontroladores PIC ... 105
Temporizador 0 (*Timer*0: TMR0) ... 106
El prescalador ... 108
El *Timer*2 ... 109
El *Timer*1 ... 110
Descripción del *Timer*1 ... 111
13 Módulos CCP (captura/comparación/PWM) ... 113
Modo de comparación ... 114
Modo de captura ... 116
Modo de PWM ... 118
Ciclo útil de la PWM ... 120
Configuración de la PWM ... 120
14 Introducción a las comunicaciones seriales ... 121
Transmisión sincrónica ... 123
Transmisión no sincrónica ... 124
Comunicaciones seriales no sincrónicas ... 124
Protocolo serial no sincrónico ... 125
El canal de comunicación ... 126
El estándar RS-232 ... 126
Características eléctricas ... 127
El código ASCII ... 128
Comunicación no sincrónica en microcontroladores ... 129
Comunicación *half-duplex* ... 131
Recepción *half-duplex* ... 131
Transmisión *half-duplex* ... 133
Aplicación de las rutinas de recepción y de transmisión no sincrónicas ... 133
Comunicación serial *full-duplex* ... 135
Ejemplo de aplicación de las subrutinas ... 138
Módulo USART dentro del *chip* ... 142
Transmisión por medio de la UART ... 144
Recepción por medio de la UART ... 144
Ejercicio de comunicación *full-duplex* ... 146
Comunicaciones seriales sincrónicas ... 146
Protocolo serial sincrónico ... 149
Dispositivos periféricos comunes en la comunicación serial sincrónica ... 149

Estándares de la comunicación serial sincrónica ... 149
El bus I2C ... 150
Bus SPI ... 150
Interfaz *Microwire* ... 150
El bus I2C ... 150
Especificaciones del bus I2C ... 151
15 Los convertidores análogo/digitales ... 155
Características del convertidor AD ... 156
Características de funcionamiento del conversor A/D ... 157
Utilización del convertidor A/D ... 159
16 El *watchdog* y el modo *sleep* ... 161
El temporizador *watchdog* (WDT) ... 162
La palabra de configuración ... 163
Operación del *watchdog* ... 164
Modo *sleep* ... 165
Despertando del modo *sleep* ... 165
17 Arquitectura de los microcontroladores Motorola ... 169
La familia 68HC08 ... 171
Arquitectura de los microcontroladores Motorola ... 173
El núcleo de procesamiento ... 173
El sistema de memoria ... 174
Los módulos periféricos internos ... 176
Asignación de pines en los microcontroladores 68HC08 ... 176
18 Los puertos de los microcontroladores HC08 de Motorola ... 177
Descripción de los pines y sus funciones ... 178
Los puertos de entrada y salida (I/O) ... 180
El puerto A ... 180
El puerto B ... 181
El puerto D ... 182
19 El oscilador y el *reset*, en los microcontroladores HC08 de Motorola ... 185
El módulo interno del oscilador (OSC) ... 186
El módulo oscilador con cristal XTAL (subfamilia 68HC08XXX) ... 186
Módulo oscilador con red RC (subfamilia 68HRC08XXX) ... 187
Señales de reloj internas ... 188
Características eléctricas de los módulos osciladores ... 189
El *reset* en los microcontroladores Motorola 68HC08 ... 189
Condiciones iniciales establecidas por medio del ciclo de *reset* ... 190
Fuentes de interrupción del *reset* ... 192
20 Organización de la memoria en los microcontroladores HC08 de Motorola ... 193
Organización de la memoria ... 194
Asignación de registros dentro del área de memoria ... 194
Tabla de vectores ... 198
Memoria de datos de lectura y escritura RAM ... 198
Memoria de programa de solo lectura ROM ... 198
Memoria ROM del programa monitor (MON) ... 198
Acceso al modo monitor ... 200
Vectorización en el modo monitor ... 201
Velocidad de comunicación del modo monitor ... 201
Formatos básicos de comunicación del modo monitor ... 201
Comandos del monitor ... 202
21 El módulo DMA y los modos de direccionamiento en los microcontroladores 68HC08 ... 203
Acceso directo a memoria (DMA) ... 204
El módulo DMA08 ... 204
Operación del módulo DMA08 ... 205
Características resumidas del módulo DMA08 ... 206
Registros de control y estado del módulo DMA08 ... 206
Los modos de direccionamiento en la familia HC08 ... 209
Direccionamiento inherente ... 210
Direccionamiento inmediato ... 210
Direccionamiento directo ... 211
Direccionamiento extendido ... 211
Direccionamiento indexado, sin desplazamiento ... 212
Direccionamiento indexado, con desplazamiento de 8 *bits* ... 212
Direccionamiento indexado, con desplazamiento de 16 *bits* ... 212
Direccionamiento del SP (*stack pointer*), con desplazamiento de 8 *bits* ... 213
Direccionamiento del SP, con desplazamiento de 16 *bits* ... 213
Direccionamiento relativo ... 213
Direccionamiento de memoria a memoria, inmediato a directo ... 214
Direccionamiento de memoria a memoria, directo a directo ... 214
Direccionamiento de memoria a memoria, indexado a directo con incremento ... 215
Direccionamiento de memoria a memoria, directo a indexado con incremento ... 215
Direccionamiento indexado con incremento posterior ... 215
Direccionamiento indexado con *offset* de 8 *bits* e incremento posterior ... 216
22 Las interrupciones en los microcontroladores 68HC08 ... 217
Procesamiento de interrupciones en la CPU08 ... 218
Reconocimiento de la fuente de interrupción ... 218
Evaluación del nivel de prioridad (arbitraje) ... 219
Almacenamiento del estado actual del microcontrolador ... 220
Enmascaramiento y anidamiento de las interrupciones ... 220
Retorno de la interrupción (RTI) ... 222
Módulo de interrupción externa (IRQ) ... 222
Pin de interrupción externa (IRQ1)
Registro de estado y control de la interrupción IRQ (INTSCR) ... 224
23 Los módulos especiales del 68HC08: COP, LVI y KBI ... 225
El módulo COP (*computer operating property*) ... 226
Funciones del módulo COP ... 227
Las señales del módulo COP ... 227
El módulo inhibidor por volatje bajo LVI (*low voltage inhibitor*) ... 228
Registros de control del módulo LVI (CONFIG2 y CONFIG1) ... 229
El módulo de interrupciones para teclado KBI ... 229
Descripción del módulo KBI ... 230
Registro de estado y control del módulo KBI (KBSCR) ... 231
Registro habilitador de las interrupciones del módulo KBI (KBIER) ... 231
24 El módulo convertidor A/D (ADC) y el SIM del 68HC08 ... 233
El módulo de conversión análoga a digital ADC ... 234
Descripción del módulo ADC ... 235
Voltaje de conversión ADCVIN ... 235
Tiempo de conversión ... 235
Modo de conversión continua ... 235
Las interrupciones del ADC ... 236
Registros de configuración del módulo ADC ... 236
El módulo del sistema de integración SIM ... 237
Fuentes internas de interrupción del *reset* ... 238
El *reset* a la conexión de la alimentación POR ... 239
El *reset* por código de operación ilegal ... 239
El *reset* por dirección ilegal ... 239
El *reset* de los módulos COP y LVI ... 239
Los registros de estado del módulo SIM ... 239
25 El módulo de la interfaz de temporización (TIM) del 68HC08 ... 241
El TIM del 68HC08 ... 242
Funciones del TIM ... 243
El modulador PWM ... 244
Registros de entrada y salida del TIM ... 244
26 Los microcontroladores BASIC Stamp ... 249
El BASIC Stamp I ... 250
Distribución de pines del BASIC Stamp I ... 251
Arquitectura del BASIC Stamp I ... 252
El regulador interno y la tarjeta de prototipos del BASIC Stamp I ... 252
Mapa de memoria disponible para el usuario ... 252
Control de los puertos del BASIC Stamp I ... 253
Programación del BASIC Stamp I ... 253
Conjunto de instrucciones resumidas del lenguaje PBASIC ... 254
27 Conjunto de instrucciones de los microcontroladores BASIC Stamp I ... 257
Conjunto de instrucciones del lenguaje PBASIC del BASIC Stamp I ... 258
28 Los microcontroladores BASIC Stamp II ... 269
El BASIC Stamp II ... 270
Distribución de pines del BASIC Stamp II ... 270
Arquitectura del BASIC Stamp II ... 271
La tarjeta de prototipos del BASIC Stamp II ... 271
Mapa de memoria del BASIC Stamp II ... 272
Configuración de los puertos I/O del BASIC Stamp II ... 272
Programación del BASIC Stamp II ... 273
Instrucciones del lenguaje PBASIC 2 ... 273
29 Los microcontroladores 8051 de Intel ... 277
Los microcontroladores Intel ... 278
Arquitectura básica de los microcontroladores 8051 ... 278
Descripción de pines del microcontrolador 8051 ... 279
Organización de la memoria en el 8051 ... 280
Memoria de programa ... 281
Memoria de datos ... 281
El reset en los MCU 8051 ... 284
30 Puertos periféricos internos del microcontrolador 8051 ... 285
Los puertos del MCU 8051 ... 286
Los temporizadores-contadores (*timers* 0 y 1) del MCU 8051 ... 288
Modos de operación de los *timers* 0 y 1 ... 288
31 Interrupciones del microcontrolador 8051 ... 291
Introducción ... 292
Las interrupciones del MCU 8051 ... 292
Vectorización de las interrupciones ... 293
Orden de prioridad de las interrupciones ... 294
Interrupciones internas de los *timer* 0 y 1 ... 296
32 Comunicaciones seriales con el microcontrolador 8051 ... 297
El UART del microcontrolador 8051 ... 298
Configuración de la velocidad de comunicación ... 298
MODO0 de comunicación ... 299
MODO1 de comunicación ... 301
MODOS 2 y 3 de comunicación ... 301
33 Los microcontroladores AVR de 8 *bits* de ATMEL ... 305
ATMEL ... 306
El microcontrolador AT90S2313 de la familia AT90 ... 306
Descripción de pines del AT90S2313 ... 306
Arquitectura del AT90S2313 ... 307
Organización de la memoria ... 307
Modos de direccionamiento de datos y programas ... 312
Estructura de los puertos de entrada y salida I/O ... 314
El *reset* y las interrupciones ... 315
El registro máscara de interrupciones generales GIMSK ... 317
El registro bandera de interrupciones generales GIFR ... 318
Diagramas de tiempos de los estados de *reset* ... 318

Obtenga su certificado de entrenamiento en sólo 40 semanas

Al final del curso se publicará un completo cuestionario para la evaluación de sus conocimientos.

Al contestarlo correctamente, usted obtendrá un certificado de estudios expedido por CEKIT S.A.

Capacítese en Microcontroladores y forme parte activa de la economía del nuevo milenio en su profesión o su hoobie, realizando este fácil y rápido...

Certificado de Aptitud

CEKIT
COMPAÑIA EDITORIAL TECNOLOGICA

Certifica que:

Manuel Suarez

Completó satisfactoriamente el curso sobre

Microcontroladores

incluyendo Teoría, Programación y Diseño.

CEKIT
Le ayudamos a construir su futuro

Director Técnico

Fecha

Ciuda

CURSO PRACTICO SOBRE

MICROCONTROLADORES

Otro producto con la calidad y garantía de

CEKIT S.A.
Capacitamos para vivir mejor

Es de hacer notar que el presente certificado da idea de haber cumplido con los conocimientos básicos de la teoría y práctica del curso.